MASTER EN SCIENCES INFORMATIQUES

Contribution à la conception et la réalisation d'un atelier logiciel orienté bases de données

Maréchal, Pierre

Award date:
1986

Awarding institution:
Universite de Namur

FACULTÉS
UNIVERSITAIRES
N. D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A LA CONCEPTION ET LA REALISATION D'UN ATELIER LOGICIEL ORIENTE BASES DE DONNEES

Mémoire présenté par
PIERRE MARECHAL
en vue de l'obtention
du titre de
LICENCIE ET MAITRE
EN INFORMATIQUE

Promoteur:
J-L Hainaut

ANNEE ACADEMIQUE 1985-1986

Nous tenons à témoigner de notre gratitude envers Monsieur le Professeur J-L Hainaut pour le grand intérêt qu'il a manifesté pour le mémoire et pour la précieuse collaboration qu'il a apportée à son élaboration et sa réalisation.

Nous remercions également Messieurs les assistants A. Delcourt, B. Van Houtte et M. Cadelli pour tous les conseils qu'ils nous ont donnés au cours de la réalisation du présent travail.

Nous adressons enfin nos plus vifs remerciements à Monsieur le Professeur Y. Pigneur qui nous a permis et aidé à réaliser dans les meilleures conditions un stage à l'Université de Lausanne.

Table des matières.

Introduction

Ce document entre dans le cadre de la conception et de la réalisation d'un Atelier Logiciel orienté Bases de Données. Derrière le concept de base de données se cache une représentation d'une partie du réel. Cette extraction du réel compose ce que l'on appelle un Système d'informations sur lequel on désire effectuer une gamme de traitements.

Dans l'ouvrage (BOD-PIG-83) on trouve une définition de ce que l'on entend par un Système d'informations; ce terme fait référence à un ensemble composé par exemple

- d'informations, représentations partielles, de faits intéressant une organisation
- de traitements, procédés d'acquisition, de mémorisation, de recherche, de communication et de transformation des informations
- de ressources humaines, techniques et organisationnelles

Le présent texte contribue à l'évolution de la démarche faite vis-à-vis d'un Système d'Informations. En effet, après l'analyse conceptuelle d'un Système, il est nécessaire de mettre en oeuvre une base de données regroupant ces informations de façon opérationnelle et efficace afin de rendre des traitements possibles.

Tout d'abord, il nous a paru important de resituer les problèmes posés par les Systèmes de gestion de Bases de Données (SGBD). Quelles que soient les réalisations concrètes effectuées dans ces différents SGBD, certains composants et principes sont communs à tous ceux-ci et doivent être reprécisés.

Ensuite, nous nous placerons dans le contexte de l'Atelier Logiciel orienté Bases de Données, développé à l'Institut d'Informatique de Namur et sur lequel nous avons été amené à contribuer dans sa conception et dans sa réalisation.

Enfin, nous développerons les deux volets principaux de notre travail qui ont été d'une part le problème posé par la portabilité de l'Atelier Logiciel et d'autre part la conception et la réalisation de certains composants du SGBD.

Chapitre 1 : Principes d'architecture des SGBD

1.1. Les fonctions des SGBD

De tout temps, l'homme a utilisé des informations, mais les moyens mis à sa disposition évoluent. L'avantage d'une Bases de Données est de rendre possible la centralisation, la coordination, l'intégration et la diffusion de l'information. Un système de gestion de bases de données est un logiciel qui permet à un utilisateur d'inter-agir avec une base de données. Sur base de (AD-DEL) et (DDSWP-82),différentes fonctions des SGBD peuvent être décrites.

1.1.1. Description

Un SGBD doit permettre une description d'un schéma de base de données. Cette description comprend un schéma logique qui est la description connue de l'utilisateur et une description physique qui est le schéma des données réellement connu du SGBD.

1.1.2. Utilisation

Dans tout SGBD, il existe une fonction d'utilisation. En effet tout SGBD doit permettre de rechercher, de sélectionner et de modifier des données.

1.1.3. Intégrité

Plus la masse d'informations est grande, plus il y a de risques que la donnée enregistrée soit erronée par rapport à la donnée réelle. Il est donc nécessaire d'introduire la notion d'intégrité d'une base de données. Cette fonction du SGBD consiste en un certain nombre de contraintes d'intégrité qui sont en fait des propriétés qui doivent toujours être vérifiées dans la base de données et cela quelles que soient les valeurs enregistrées.

1.1.4. Confidentialité

Une des fonctions d'un SGBD est d'assurer la confidentialité des informations. Toute information ne peut être communiquée à tout utilisateur. Il est donc indispensable de gérer les droit d'accès des utilisateurs aux informations d'une base de données

1.1.5. Concurrence d'accès

Lorsqu'on traite des SGBD, on doit se préoccuper de la concurrence d'accès. Cette fonction permet de résoudre les conflits d'accès parce que les programmes d'applications des utilisateurs accèdent aux mêmes informations d'une base de données.

1.1.6. Sécurité de fonctionnement

Une dernière fonction d'un SGBD est d'assurer une sécurité de fonctionnement. En effet, suite à un incident survenu au logiciel ou au matériel, un SGBD doit assurer le redémarrage du système. On définira un point de reprise qui permettra au SGBD de retrouver un état cohérent et défini. L'utilisateur doit donc pouvoir disposer des données sans que celles-ci ne soient perturbées par un incident quel qu'il soit.

1.2. Les principaux composants des SGBD

Après avoir analysé les différentes fonctions d'un SGBD, nous allons décrire un certain nombre de composants qui assurent la réalisation possible des fonctions.

1.2.1. Compilateur de schémas

Pour pouvoir manipuler des données, il est nécessaire qu'elles soient structurées et organisées entre elles sous forme d'un schéma. Ce schéma devra être connu du SGBD pour que les données puissent être introduites et manipulées. Ce schéma pourra être visualisé de façons différentes. Lorsque l'utilisateur fourni un schéma au SGBD, il faut que sa description respecte un certain nombre de contraintes, on pourra dire que le schéma est "lisible" par le SGBD.

On peut à ce stade distinguer trois types de schéma. Premièrement, il existe le schéma conceptuel qui est issu de l'analyse conceptuelle et dont le but est de disposer d'une description des données se rapprochant le plus du système réel d'informations. Deuxièmement, on dispose du schéma externe des données qui est le schéma visible par l'utilisateur. Enfin, il y a le schéma interne des données qui est la représentation du schéma externe en un schéma visible par le SGBD. Le schéma interne devra donc être déduit du schéma externe des données.

1.2.2. Compilateur de programmes d'applications

Comme l'on désire réaliser un certain nombre de traitements sur les données du SGBD, il faut permettre à différents programmes d'application de manipuler les données. Suivant les SGBD, différentes opérations seront ou non acceptées. Classiquement, on permet à l'utilisateur de pouvoir créer, supprimer, consulter ou mettre à jour des données.

1.2.3. Interpréteur de commandes

Lorsqu'on désire manipuler un SGBD, il faut formuler ce que l'on appelle une requête. Celle-ci devra être analysée pour juger de sa validité puis si elle est correcte, un traitement sera effectué et des informations seront éventuellement fournies suivant le type de requête posé. C'est le rôle de l'interpréteur de commandes d'effectuer ces opérations dont une spécification plus complète sera donnée par la suite.

1.2.4. Processeur de requêtes interactives

Une autre possibilité d'interrogation d'une base de données est fournie à l'utilisateur par le traitement interactif. L'utilisateur ne doit plus construire un programme d'application mais peut "dialoguer" avec un SGBD. Il est certain que toute question posée au SGBD sera l'objet d'une analyse afin d'en vérifier la validité. Suite à cette validation, le processeur pourra alors apporter une réponse à la requête formulée par l'utilisateur.

Ces requêtes sont les mêmes que celles formulées dans les programmes d'application. La seule différence est ce "dialogue" établi entre l'utilisateur et le SGBD puisqu'à chaque requête, l'utilisateur recevra une réponse et éventuellement des données puis le processeur attendra la requête suivante. Dans un programme d'application, on ne demande habituellemnt par de résultats intermédiaires mais des informations fournies par le SGBD sont directement réutilisées pour d'autres traitements.

Les requêtes concernent soit le schéma d'une base de données soit les données elles-mêmes relatives à ce schéma. Les requêtes sont habituellement formulées dans un langage proche du langage naturel.

1.2.5. Composants annexes

D'autres composants sont nécessaires afin de réaliser différentes fonctions.

Citons tout d'abord le générateur de rapports qui est un outil très intéressant puisque son but est de fournir à l'utilisateur une liste structurée d'informations contenues dans la base de données. Différentes présentations peuvent être envisagées en fonction de l'usage que l'on veut donner aux rapports. Suivant l'utilisation, différentes personnes peuvent être intéressées par des rapports mais ne désirent peut-être pas une même présentation des informations. Tous ces cas devront être envisagés lors de la réalisation du générateur. Deux types de rapports pourront être construits. D'une part un ensemble prédéfini de rapports qui sont standardisés et qui pourront être exécutés quel que soit le schéma de la base de données. D'autre part, on pourrait permettre à l'utilisateur de définir lui-même les rapports. Il construira ainsi des rapports contenant les informations strictement nécessaires à l'utilisation que l'on veut en faire.

Un autre composant très important est le chargeur-déchargeur de données. Lorsque des données sont exploitées, il se peut que le schéma auquel elles se rapportent, doive subir des modifications. Afin de pouvoir modifier un schéma de données tout en pouvant réutiliser les données mémorisées, il est nécessaire de décharger les données puis de les recharger sur base du nouveau schéma. Si on ne dispose pas de cet outil, les modifications du schéma que l'on pourra autoriser seront souvent très limitées afin de préserver l'intégrité des données.

Parmi d'autres composants, on peut citer le générateur d'écrans. Au départ d'informations stockées dans la base de données, il est possible de générer des écrans reprenant les données sous une forme structurée.

Pour terminer cette liste non exhaustive des différents composants d'un SGBD, nous pouvons citer un réorganisateur de données. Sa fonction est de permettre une restructuration des données dans une base. On peut par exemple désirer trier des informations selon une certaine séquence.

1.3. L'interpréteur de commandes

Dans la description des principaux composants du SGBD, nous avons signalé l'importance du rôle de l'interpréteur de commandes. Nous allons spécifier maintenant les aspects externes et internes ainsi que l'architecture de cet interpréteur.

1.3.1. Aspects externes

Pour qu'un utilisateur puisse correctement formuler les questions à un SGBD, il est impératif de connaître trois éléments.

Premièrement, le modèle de données utilisé par le SGBD doit être connu. Tout SGBD dispose de son propre modèle de données qui indique les structures avec lesquelles le SGBD fonctionne. Toute question devra être formulée sur base de ce modèle de données.

Deuxièmement,l'utilisateur doit connaître la liste des primitives traitées par le SGBD. Cette liste constitue le seul moyen de manipuler les informations d'un SGBD. C'est en fait une série de requêtes qu'il est possible de poser. Pour chaque primitive, il sera précisé la liste des arguments à fournir ainsi que les codes de retour. Les arguments permettront de préciser sur quelles données on désire que le traitement soit effectué. Les codes de retour indiqueront à l'utilisateur la façon dont s'est passé le traitement c'est-à-dire s'il a comporté des problèmes et dans l'affirmative la nature de ceux-ci.

Troisièmement, les règles d'enchaînement doivent être fournies. En effet, une liste des primitives est insuffisante pour l'utilisateur s'il ne connaît pas les règles d'enchaînement de ces requêtes. Il est clair par exemple qu'une lecture d'une information contenue dans un fichier ne pourra être faite que si ce fichier a été précédemment ouvert.

1.3.2. Aspects internes

1.3.2.1. Fonctions internes

Lorsqu'un appel est formulé à l'interpréteur de commandes, un certain nombre d'opérations sont effectuées entre l'appel à la fonction et l'expression de la réponse à cette requête.

Tout d'abord, la commande doit être validée. Il est en effet nécessaire de vérifier que la commande existe, que le nombre d'arguments est correct et que les arguments eux-mêmes sont valides. Par exemple, si un des paramètres d'une fonction est déclaré comme un pointeur, on passera l'argument par adresse ou par valeur suivant la spécification. Si une erreur existe au niveau de l'appel, il ne servira à rien de continuer le traitement, un code de retour indiquera à l'utilisateur le type d'erreur qui a été commis.

Lorsque la commande a été validée, il faut consulter le schéma pour lequel une requête a été émise. La transformation du schéma externe en un schéma interne permet au SGBD d'avoir une représentation du schéma du SGBD et donc de pouvoir le consulter.

Pour pouvoir accéder aux informations relatives à un schéma du SGBD, il faut positionner les courants des fichiers. Un courant est le dernier article manipulé dans une séquence déterminée. Par le positionnement correct des courants, on assure la correspondance entre les informations demandées et celles qui sont fournies. Le problème des courants des fichiers est délicat dans la mesure où il n'est pas géré de la même façon dans chaque système. Nous verrons dans la cadre de la réalisation une application concrète de ce problème de gestion de courants dans deux sytèmes.

Si un appel a été formulé à une fonction de l'interpréteur de commandes, il l'a été suivant une certaine sémantique. Mais au niveau des fichiers réels contenant les informations, l'appel initial ne peut pas être compris. Il faut donc traduire les termes utilisés pour l'appel de la fonction de l'interpréteur de commandes en termes compréhensibles par le SGBD réel. On dira qu'on traduit les appels des primitives de haut niveau en appels aux primitives physiques.

Lorsque la traduction est effectuée, le SGBD réel est capable de comprendre la requête posée et donc d'activer ses propres mécanismes pour y répondre. Nous appelons ici mécanismes, les accès élémentaires qui sont effectués sur un fichier c'est-à-dire l'ouverture,la fermeture d'un fichier, la lecture, l'écriture, la mise-à-jour, la suppression d'un enregistrement.

Toute exécution doit être contrôlée c'est-à-dire que tout traitement même élémentaire doit faire l'objet de vérifications qui indiquent la façon dont l'opération s'est effectuée. Par exemple, l'écriture d'un enregistrement dans un fichier ne peut se faire que si le fichier a été ouvert. Si on omet l'ouverture du fichier, l'opération ne pourra être traitée et un code d'erreur devra être répercuté jusqu'à l'utilisateur pour lui signaler la faute dans l'enchaînement des requêtes.

Quelle que soit la façon dont le traitement s'est effectué, l'interpréteur devra clôturer ses opérations par la construction d'une réponse. Si le traitement ne s'est pas bien déroulé, la réponse sera très simple et sera formulée le plus souvent par un code d'erreur indiquant le type de faute commise. Si tout s'est bien passé, l'interpréteur fourni soit un code indiquant que l'opération demandée s'est bien déroulé soit un ensemble de valeurs correspondant à la requête émise. Il arrive que ces deux possibilités soient combinées.

1.3.2.2. Architecture

Nous nous sommes basés sur (MAR-75) pour fournir ci-après la liste des opérations que l'on retrouve dans tout système de gestion de base de données (DBMS). L'architecture cache l'ensemble des mécanismes nécessaires au bon fonctionnement de ces opérations.

Premièrement, un programme d'application fait appel au DBMS pour lire un enregistrement. Suite à cet appel, le DBMS consulte la description logique des données pour connaître les types de données logique nécessaires. Ayant déterminé ces informations, le DBMS examine la description physique de la base de données et détermine les enregistrements physiques à lire. Le DBMS envoie une commande au système d'exploitation pour qu'il lise les informations en question. Les données demandées sont transférées de la mémoire vers les buffers. Disposant des informations, le DBMS dérive l'enregistrement logique nécessaire pour le programme d'application puis transfert les données depuis les buffers vers l'espace de travail du programme d'application. Le DBMS fournit, en plus des informations, l'état de celles-ci, incluant les types d'erreurs rencontrés. Suite à ces opérations, le programme d'application peut travailler avec les données disponibles dans son espace de travail.

L'importance d'une bonne hiérarchie ne se retrouve pas uniquement dans le fait qu'elle est plus facilement développable mais sutout dans sa plus grande facilité de maintenance.

Signalons enfin que dans la conception de cette hiérarchie, on retrouve divers niveaux de programmation qui correspondent à divers niveaux d'indépendance. En fait, on peut distinguer deux types d'indépendance entre les données et les programmes. Premièrement une indépendance physique qui permet de modifier l'organisation physique des données sans modifier les programmes d'application. Deuxièmement une indépendance logique dont le but est d'accepter un changement du schéma conceptuel des données sans devoir changer les programmes d'application.

1.4. Gestion des définitions de données

Lorsqu'on traite de la gestion des définitions de données, il est nécessaire de distinguer plusieurs niveaux de perception. Nous nous sommes basés sur (DDSWP-82) pour définir ces différentes notions des définitions de données.

Premièrement, il y a la vision conceptuelle. Le schéma qui en résulte provient de l'analyse conceptuelle dont le but est de fournir une description dénuée de toute caractéristique technique mais qui soit une représentation correcte et naturelle du système réel. Cette description est donc indépendante de l'utilisation des données et des paramètres techniques de la base de données.

Ensuite nous devons présenter la vision logique. Une description sous cette forme reprend les spécifications conceptuelles enrichies de la spécification des chemins d'accès aux données nécessaires aux applications. Les données sont représentées sous une forme permettant l'exécution d'applications par une machine abstraite.

La troisième représentation concerne la vue externe des définitions de données. Cette perception est celle des utilisateurs de la base de données. Il est habituel de disposer d'un ensemble de schémas externes dans la mesure où chaque groupe d'utilisateurs peut avoir une vue différente de la base de données.

La dernière perception que nous citons est une vue interne des définitions de données. En effet, cette représentation concerne les objets techniques stockés sur des supports de mémoire et dont les caractéristiques visent à optimiser les consommations de ressources lors de l'exploitation de la base de données. Toute modification d'un paramètre technique entraîne donc un changement du schéma interne.

Ayant examiné les différentes visions possibles d'un dictionnaire de données, nous pouvons déduire trois formes de celui-ci.

La première forme est le schéma source qui constitue l'entrée pour le compilateur de schémas décrit précédemment.

La deuxième forme est la version compilée du schéma. Celle-ci peut se diviser en deux parties avec d'un côté le schéma externe, et de l'autre le schéma interne.

La troisième forme est la version enregistrée. Cette version constitue ce que l'on appelle le schéma du dictionnaire de données.

En fait un dictionnaire de données est un catalogue standard et organisé, centralisant les informations sur un système d'informations. Le rôle d'un dictionnaire de données doit être examiné dans l'organisation d'une entreprise. Chacune d'elle dispose de diverses ressources: financières, en personnes, physiques, en informations. Un élément essentiel au bon fonctionnement d'une entreprise repose dans la qualité de ses informations qui doivent être précises et à jour. Malheureusement, beaucoup d'entreprises n'ont pas encore réalisé l'importance d'une gestion de l'ensemble des informations. C'est dans ce cadre de description d'un système d'informations qu'il faut inclure la notion de dictionnaire de données.

On peut adopter plusieurs comportements vis-à-vis d'un dictionnaire de données.

Tout d'abord on peut accorder un rôle passif à un dictionnaire de données en lui donnant comme fonction de documenter l'utilisateur sur son système c'est-à-dire de l'ensemble des renseignements relatifs à l'organisation du système d'informations.

Ensuite nous pouvons examiner le rôle actif d'un dictionnaire de données. Ce rôle existe si le dictionnaire fournit, outre de la documentation, une aide pour la création de nouvelles structures de données ou applications. Divers outils d'aide au développement d'un projet peuvent ainsi être utilisés: outils d'analyse d'une structure de données, d'évaluation du contenu d'une structure de données, de traduction de structure en rapports graphiques, d'analyse de l'usage des données et de leur traffic, de test de cohérence et d'intégrité, d'analyse de l'impact d'un changement sur l'organisation,etc.

Enfin, un rôle dynamique peut être assigné à un dictionnaire de données. Ce rôle consiste en une interaction directe avec les programmes pendant leur exécution; ceci permettant par exemple à un programme de connaître la localisation d'une information et d'en utiliser les valeurs.

1.5. Les dictionnaires de données.

La partie ci-après est consacrée à l'analyse de trois dictionnaires de données. Premièrement nous nous intéressons au dictionnaire de données IDD (Integrated Data Dictionary) de Cullinet, ensuite nous traitons du CDD (Common Data Dictionary) qui est le dictionnaire de données développé sur le système Vax) et enfin nous examinerons le système ORACLE.

Pour construire ces différents cas nous avons utilisés (BOV-85), (CUR-DIEC), (DD-84) et (MA-82).

1.5.1. IDD de Cullinet

L'IDD constitue le noyau du SGBD de Cullinet. Par l'intermédiaire d'interfaces spécialisés, l'IDD reçoit des informations fournies par divers éléments du système. Parmi ces éléments, nous pouvons citer:

- IDMS/CULPRIT qui est un outil de recherche d'informations qui génère des rapports à partir de la base de données et de fichiers conventionnels
- IDMS qui permet la gestion de la base de données
- DDS est un outil de gestion de bases de données réparties
- ADS est un outil de recherche et de mise-à-jour dans IDMS
- INTERACT est un outil d'édition de texte, de développement de programmes et de documentation
- ON-LINE-QUERY est un outil d'interrogation de la base de données en langage quasi naturel

L'organisation de IDD permet de gérer manuellement ou automatiquement des informations concernant:

- les sources des données
- les relations entre les données
- les caractéristiques physiques des données
- les fréquences d'accès
- les autorisations d'accès
- la localisation des données
- ...

qui seront accessibles sous forme de rapports.

Afin d'analyser l'organisation de IDD, on peut distinguer trois sortes d'entités:

a) une ensemble d'entités qui correspondent aux composants des systèmes de gestion de l'information
ex: utilisateur, système, fichier, enregistrement, module, programme,...

b) des entités correspondant à l'environnement du système de gestion des données
ex: écran, tâche, file d'attente, terminal, ligne,...

c) un certain nombre d'entités spéciales qui permettent, en gérant du système, de créer de nouveaux types d'entités.

La gestion du dictionnaire se fait via un langage nommé DDDL dont les requêtes sont soumises on-line ou en batch.

1.5.2. CDD sur système VAX

Le CDD est le dictionnaire de données du VAX. Dans celui-ci ne se trouvent pas les notions de programmes, d'utilisateurs, de fichiers et de relations. Son but est de centraliser des définitions de données dans un but de cohérence et de contrôle de la redondance. A chaque création d'une nouvelle base de données, les deux gestionnaires de bases de données sur VAX (VAX-11 Datatrieve et VAX DBMS) rangent lors de la compilation les définitions des constituants de la base de données dans le CDD.

Le dictionnaire est organisé de façon hiérarchique, sous forme d'arbre; chaque noeud n'ayant qu'un seul parent. On distingue deux types de noeuds: les "dictionary directory" et les "dictionary object".

Un utilisateur accède alors au CDD via le DATATRIEVE ou le DBMS ou par accès direct. L'assignation se fait à la racine de l'arbre sauf si l'utilisateur précise le noeud souhaité. De plus, à chaque noeud correspond une ACL (access control list) définissant les droits des utilisateurs ainsi qu'une HISTORY LIST rapportant les manipulations effectuées sur ce noeud.

La hiérarchie permet donc à l'utilisateiur de gérer sa partie du dictionnaire selon ce qu'il désire réaliser dans les limites des contraintes du système.

Pour utiliser le dictionnaire, l'utilisateur dispose de trois outils:

- premièrement, le DATATRIEVE est un outil de gestion très sophistiqué qui permet
a) la mise en place et l'exploitation de bases de données relationnelles
b) une connexion avec les bases de données organisées par le DBMS
c) une manipulation du contenu des données
d) un utilitaire d'impression des requêtes
e) un utilitaire de gestion d'écran
f) un utilitaire pour créer des graphiques
g) des possibilités d'accès aux données interactivement ou via des programmes d'application
h) des possibilités d'accès via un réseau DEC-NET

- deuxièmement, le VAX DBMS est l'équivalent du CODASYL; l'organisation des données se fait sous forme de record, item, set, area, ...

- troisièmement, le DMU est un utilitaire permettant l'organisation de la structure du dictionnaire de données c'est-à-dire de toutes les informations reprises dans les HISTORY LISTs et dans les ACLs.

1.5.3. ORACLE data dictionary

L'ORACLE data dictionary est constitué d'un ensemble de tables reprenant des informations sur une base de données, ce dictionnaire étant créé à chaque création d'une base de données. Les tables du dictionnaire de données peuvent être lues via des requêtes SQL.

liste des tables décrivant le dictionnaire:

- DTAB description des tables et des vues du dictionnaire (une vue est une partie d'une table)
- SYSCATALOG structure des tables et des vues accessibles par les utilisateurs
- CATALOG structure des tables accessibles par l'utilisateur
- TAB liste des tables, vues, cluster, synonymes créés par l'utilisateur (un cluster est un groupe d'informations)
- SYSCOLUMNS spécifications des colonnes accessibles dans les tables et les vues
- COLUMNS spécification des colonnes des tables
- COL spécification des colonnes des tables créées par l'utilisateur
- SYSINDEXES liste des index, soulignant les colonnes, le créteur et les options
- SPACES sélection des espaces de définitions pour la création des tables et des clusters
- VIEWS liste des phrases SQL sur lesquelles les vues sont basées
- SYSTABAUTH directory des autorisations d'accès accordés par ou pour l'utilisateur
- EXTENTS structure de données des extensions dans les tables
- STORAGE données et allocation de mémoire d'index pour les utilisateurs propriétaires des tables
- SYSSTORAGE résumé de toutes les bases de données mémorisées
- SYSUSERAUTH liste des utilisateurs d'ORACLE
- SYSEXTENTS structure de donnée des tables à l'intérieur du système
- PARTITIONS structure des fichiers avec leurs partitions

tables décrivant d'autres tables:
la table TAB contient l'ensemble des noms et descriptions de toutes les tables,vues,synonymes et clusters créés. Par exemple, on peut décrire un élément de type employé, salaire, département, utilisateur,...

tables décrivant les privilèges d'accès:
ces tables permettent à chaque utilisateur de dire qui peut avoir accès aux informations (en détaillant les types d'informations) et quelles opérations sont acceptées pour cet utilisateur.

tables décrivant les colonnes:
pour chaque table, cela permet de décrire les élèments qui la compose ainsi que les différentes valeurs possibles.

tables décrivant les vues:
ces tables donnent la possibilité d'exprimer les vues possibles d'une table. En effet une vue étant un sous-ensemble d'une table, cela évite que tous les utilisateurs accèdent à toutes les informations d'une table.

tables décrivant la structure physique d'une base de données:
ces tables permettent par exemple de citer les index

Chapitre 2 : Le contexte de l'atelier logiciel orienté bases de données

2.1. Description générale,objectifs et principes

L'atelier logiciel orienté bases de données est un outil indépendant mais il peut également entrer dans le cadre du projet IDA.

L'atelier est constitué des éléments classiquement décrits qui ont été présentés dans la première partie de ce document; l'atelier comprend un interpréteur de commandes ainsi qu'un certain nombre d'autres composants utiles à la manipulation des données, l'autre partie principale est la gestion des définitions de données dont le but est de gérer les schémas de données relatifs aux systèmes d'informations que l'on désire exploiter.

Comme précisé précédemment, l'atelier logiciel orienté bases de données qui nous a occupé, peut aussi être vu comme un produit intégré au logiciel IDA dans le but principal d'une décentralisation. Le logiciel IDA, réalisé à l'Institut d'Informatique des Facultés Universitaires de Namur en coopération avec le projet ISDOS développé à l'Université de Michigan, est un système-logiciel d'aide à la conception d'un système d'informations. Le coeur du système IDA est la bases de données des spécifications qui regroupe l'ensemble des spécifications du système d'informations introduites et rédigées à l'aide d'un langage nommé DSL. Divers outils ont été créés afin d'exploiter cette base de données. C'est ainsi qu'il existe un langage interactif d'interrogation de la base de données ainsi que la possibilité d'éditer un ensemble de rapports documentaires et d'analyse présentant différents aspects du système spécifié. D'autre part, un générateur automatique d'un programme de simulation ainsi qu'un générateur de maquette ont été développés.

Si l'on considère IDA comme base de données centrale, l'atelier contient la base de données locale. Comme le schéma ci-après le montre, l'objectif est de pouvoir retirer un certain nombre d'informations au départ de la base de données centrale et de les introduire dans la base de données locale. Ensuite, une série d'outils permettrait de manipuler ces données au niveau de la base de données locale. Lorsque ces informations seront jugées correctes, il sera possible de les réintroduire dans la base de données centrale. Rappelons qu'il est toujours possible d'introduire des informations dans la base de données centrale sans qu'elles proviennent de la base de données locale.

En ce qui concerne les outils qui traiteront la base de données locale, l'accent est d'abord mis sur des outils spécifiques (transformation de schémas, vérificateur de conformité, générateur de code, saisie et consultation,...). Ensuite, des outils plus généraux vont être développés (consultation interactive, mise-à-jour interactive, générateur de rapports...).

La définition ci-dessus des différents outils développés pour la base de données locale montre le but principal de l'atelier. Celui-ci prend principalement en charge les phases de conception logique et physique de la base de données reprises de la démarche de conception (HAI-86). La conception logique permet au départ d'une solution conceptuelle de construire une solution correcte, efficace et strictement indépendante des machines réelles. La conception physique consiste à produire une solution correcte, efficace mais qui cette fois soit exécutable par une machine réelle.

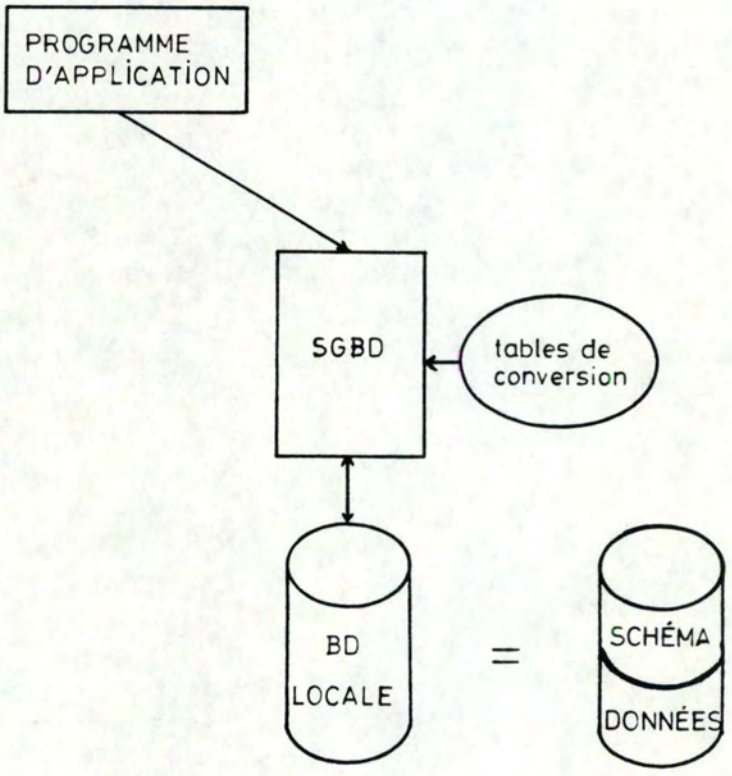
Deux avantages fondamentaux de cet atelier peuvent être décrits par rapport à d'autres logiciels.

D'une part, il existe une totale indépendance du logiciel. Quel que soit le SGBD que l'on utilise, l'atelier pourra y être facilement adapté. La meilleure preuve est qu'actuellement, l'atelier logiciel orienté bases de données est manipulable avec des fichiers RMS mais également avec des fichiers d'un Dbase 3 de Lattice. Tout autre système pourra être utilisé car les modifications à apporter au logiciel sont maintenant parfaitement contrôlées et ne demandent qu'un travail limité d'adaptation.

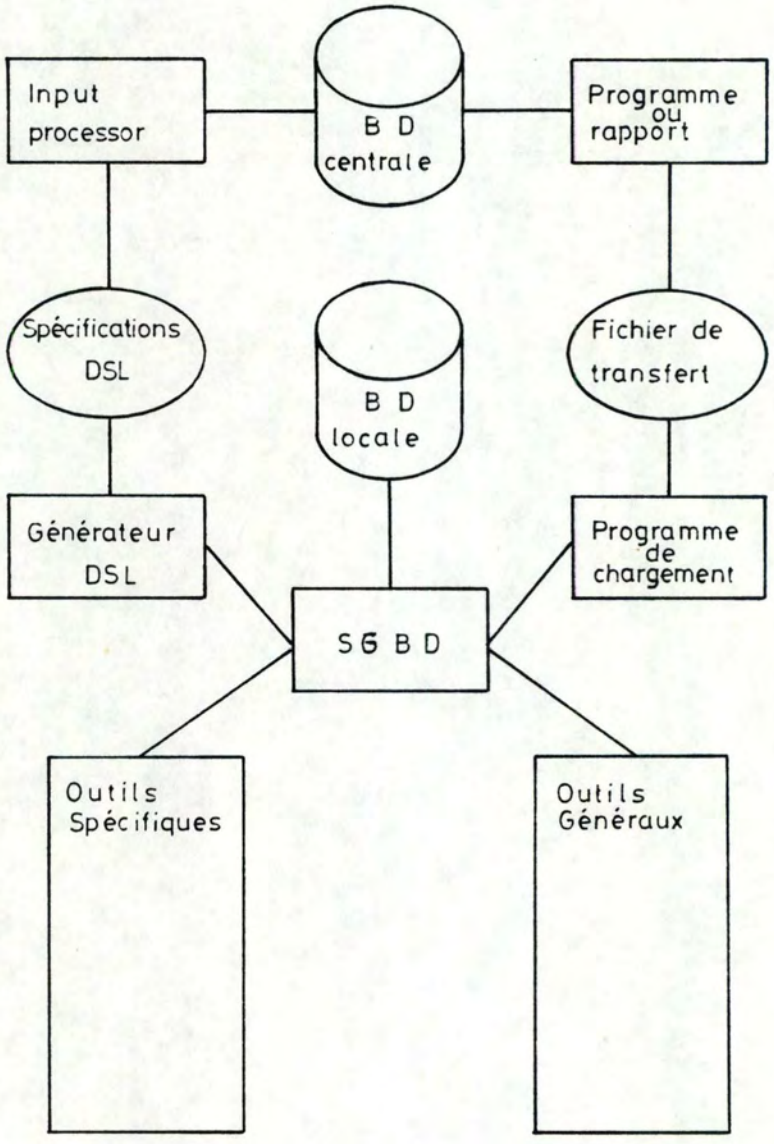
D'autre part, le logiciel dispose d'une portabilité très grande. En effet, actuellement l'atelier est exploitable sous mini-ordinateur mais également sous micro-ordinateur. Initialement, l'atelier a été conçu sur mini-ordinateur avec pour système de fichiers un RMS. Par la suite, le logiciel a été transporté sur micro-ordinateur en utilisant un système Dbase 3 développé par Lattice.

Ces deux avantages assurent au logiciel une très grande diffusion possible puisque quel que soit le système de fichiers ou quel que soit la gestion des données, l'atelier peut être facilement adapté.

Lorsqu'on examine le schéma général de l'atelier décrit ci-après, on y constate un élément appelé "tables de conversion". Celles-ci sont à mettre en rapport avec les différentes notions de schéma vues précédemment. Ces tables permettent de transformer les éléments du schéma externe des données en des composants du schéma interne. Elles constituent en fait une traduction d'éléments connus de l'utilisateur en composants connus uniquement de façon interne et permettant la manipulation des données d'une base.



ARCHITECTURE LOCALE



ARCHITECTURE GENERALE

2.2. Le SGBD de l'atelier

Les fonctions du SGBD de l'atelier remplissent les conditions décrites de façon générale dans la première partie.

Le principaux composants ont pour certains déjà été développés alors que d'autres sont en cours de développement.

Tout d'abord c'est l'interpréteur de commandes qui a été réalisé. Il constitue l'élément le plus important au niveau du codage de l'atelier. C'est lui qui assure à l'utilisateur le fait de pouvoir agir sur des schémas et les données de toute base puisqu'il réalise l'interface entre les programmes d'application et la base de données.

Après cette réalisation, un certain nombre d'outils ont été développé en parallèle. C'est le cas d'abord du chargeur qui permet au départ d'une description dans un langage défini, de charger une base de données de l'atelier.

Ensuite, un générateur de rapports a été réalisé. Celui-ci permet à l'utilisateur de recevoir les données d'une base sous une forme structurée et qui l'informent d'un certain nombre de caractéristiques de celles-ci.

Un autre outil très important est le vérificateur de conformité. Son but est de vérifier qu'un schéma est conforme au SGBD de l'atelier, pour ce faire un certain nombre de règles doivent être vérifiées.

Enfin, nous citons ici le générateur qui fait partie du travail présenté et dont les spécifications complètes seront données par la suite. Ce générateur s'occupe de créer les tables de conversion d'un schéma de données (ces tables sont parfois appelées tables internes). De plus il permet de créer les tables externes du schéma. Les informations contenues dans ces tables ne sont pas encore exploitées à ce jour mais dans l'avenir seront utilisées par un définisseur d'écran, par un analyseur syntaxique et par sans doute d'autres produits non encore définis. La dernière fonction du générateur est de créer un ensemble d'informations relatives à un schéma de données et qui constituent un environnement pour le programmeur.

2.3. L'interpréteur de commandes

2.3.1. Aspect externe

Lorsqu'on traite d'un SGBD, on se base toujours sur un modèle pour décrire les structures et les données. Dans le cas présent, c'est un sous-ensemble du modèle MAG (modèle d'accès généralisé) qui est utilisé. Sa connaissance permet de comprendre les primitives décrites dans l'interpréteur de commandes ainsi que les schémas permettant la description des données.

2.3.1.1. Le modèle d'accès généralisé

Le modèle d'accès généralisé (MAG) est un relevé des concepts essentiels que l'on retrouve sur les systèmes de gestion de données sur mémoires secondaires et cela aussi bien pour les systèmes de gestion de bases de données (SGBD) que pour les systèmes de gestion de fichiers (SGF).

Ce modèle permet de décrire d'une part les structures de données d'un point de vue sémantique et d'autre part les accès possibles sur ces structures.

Une présentation complète de ce modèle se trouve dans (HAI-86). Nous ne reprenons ici qu'un résumé des concepts principaux utilisés dans l'atelier.

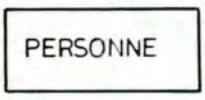
article et type d'article
- - - - - - - - - - - - - - -

article ou record : entité stockée d'information qui peut être créée et supprimée et à laquelle on peut accéder

type d'article ou record-type : regroupe 0 ou plusieurs articles et en défini les propriétés communes

exemple

PERSONNE désigne un type d'article. Toute personne décrite par ce type d'article est un article de ce type. Le type d'article est représenté comme suit:



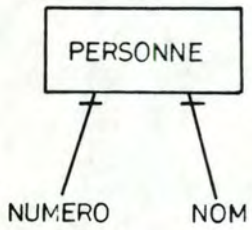
item obligatoire et valeur d'item
- - - - - - - - - - - - - - - - - - -

valeur d'item : élément associé à un record, à tout instant 0,1 ou plusieurs valeurs d'items sont associées à un article

item obligatoire : à un type d'article, on peut associer 0,1 ou plusieurs valeurs appelées items. Tout item est obligatoire c'est-à-dire qu'à tout article est associé au moins une valeur de cet item.

exemple

NUMERO et NOM sont des items. La représentation permet de distinguer à quel type d'article un item est associé. Une valeur d'item sera par exemple DUPONT pour le NOM. Le caractère obligatoire de l'item est représenté par le court trait continu, ainsi NUMERO et NOM sont des items obligatoires pour PERSONNE.



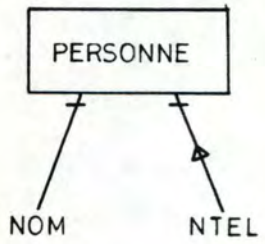
item simple ou répétitif

item simple : à chaque article on ne peut associer qu'une seule valeur de cet item

item répétitif : à un record on peut associer plus d'une valeur de cet item

exemple

NOM est un item simple alors que NTEL est un item répétitif. La représentation appliquée par le triangle indique le caractère répétitif et indique le sens de la répétitivité; c'est à une personne que peuvent correspondre plusieurs numéros de téléphone.



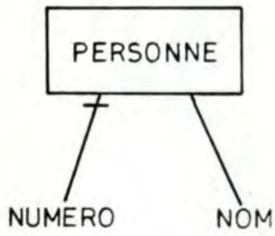
item identifiant ou non identifiant

item identifiant : pour toute valeur de cet item, il n'existe pas plus d'un article associé à cette valeur

item non identifiant : contraire de l'item identifiant

exemple

NUMERO peut être identifiant de PERSONNE alors qu'un NOM n'est sûrement pas identifiant



chemin d'accès, type de chemin d'accès, origine et cible

chemin d'accès ou path : mécanisme qui associe à un record origine une suite de 0,1 ou plusieurs articles cibles, de plus à partir de l'origine on peut accéder aux articles cibles.

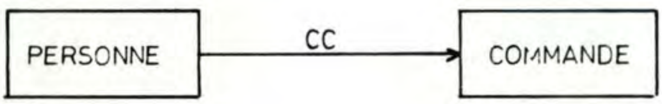
type de chemin d'accès ou path-type : tout chemin appartient à un et un seul type de chemin qui en défini les propriétés communes

origine : l'origine représente le record au départ duquel on peut parcourir un chemin

cible : la cible représente le record terminant un chemin

exemple

CC est un type de chemin entre PERSONNE et COMMANDE. Le mécanisme qui associe par exemple à une personne toutes ses commandes sera le chemin d'accès. Sa représentation s'effectue par la flèche. PERSONNE est donc l'origine du chemin et COMMANDE la cible. Les flèches peuvent êtres omises s'il existe un chemin de PERSONNE vers COMMANDE et de COMMANDE vers PERSONNE.

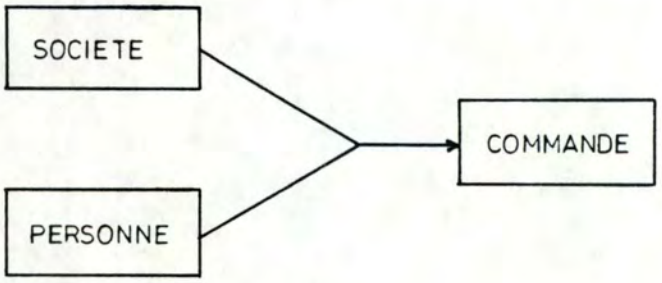


type de chemin multi-origines
- - - - - - - - - - - - - - - - -

type de chemin multi-origines : type de chemin ayant plusieurs origines

exemple

SOCIETE et PERSONNE constituent les origines du chemin qui est don multi-origines.

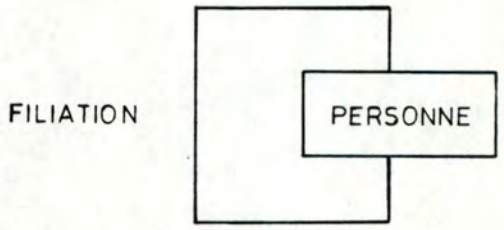


type de chemin récursif
- - - - - - - - - - - - - -

type de chemin récursif : un même type d'articles est à la fois cible et origine

exemple

Par le type de chemin FILIATION on peut dire que PERSONNE est origine et cible et que donc FILIATION est un type de chemin récursif



classe fonctionnelle
- - - - - - - - - - - - -

classe fonctionnelle : caractérise le nombre maximum d'articles d'un membre que l'on peut associer à chaque article de l'autre membre

1-N : un chemin contient un nombre quelconque de cibles alors qu'une cible ne l'est que d'un seul chemin
représentation : —<—

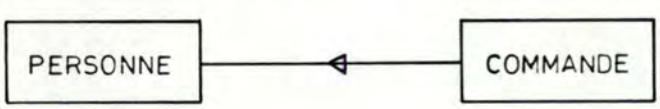
N-1 : un chemin contient une seule cible mais celle-ci peut-l'être d'un nombre quelconque de chemins
représentation : —>—

1-1 : un chemin contient une seule cible et inversément
représentation : ———

N-N : un chemin contient un nombre quelconque de cibles et une cible peut l'être d'un nombre quelconque de chemins
représentation : —>—<—

exemple

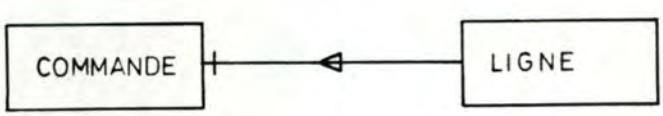
A une personne peut être associée plusieurs commandes mais à une commande appartient à une seule personne



contrainte d'existence
- - - - - - - - - - - - - -

contrainte d'existence : consiste à imposer à tout article d'un membre d'un type de chemin d'être associé à au moins un article de l'autre membre. La représentation de cette contrainte se fait par un trait court continu tracé perpendiculairement à la ligne indiquant un type de chemin.

exemple

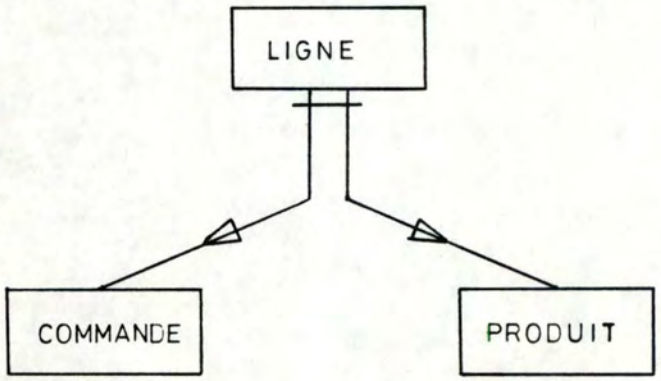


identifiant composé

identifiant composé : notion d'identifiant étendue à plusieurs types d'articles associés au type d'article identifié

exemple

Cette représentation indique que toute ligne est identifiée par une commande et un produit.



2.3.1.2. Interface ADL-C

Une des façons d'accéder aux informations d'un SGBD est d'utiliser les primitives appelées primitives ADL-C. L'ADL-C est une extension du langage C par des instructions du langage ADL défini dans le cadre de la démarche de conception de base de données développé à l'Institut d'Informatique. Les informations sur l'interface ADL-C sont basées sur (CAD-86).

Ces primitives ADL-C ont été construites pour faciliter la programmation. En effet, l'architecture de l'atelier ne présentait initialement qu'une interface à un seul point d'entrée. L'accès s'effectuait par une seule primitive contenant plusieurs arguments mais qui n'étaient jamais tous exploités en même temps. Les primitives ADL-C permettent de mieux distinguer le type d'appel effectué puisque des noms significatifs ont été donné à chaque primitive et de ne spécifier que les arguments nécessaires à chaque fonction.

Un certain nombre d'objets sont disponibles pour le programmeur:

Tout d'abord des constantes de codes. Ceux-ci se présentent sous forme de constantes dont les définitions (DEFINE) ont été générées automatiquement. Ces codes sont des entiers et appartiennent à trois catégories : les codes des types d'articles (ex: CLIENT), les codes des types de chemins (ex : CCOM) et les codes d'erreurs. De plus,un code spécial RECORD dont la valeur est 0 (zéro) signale qu'un type d'article est inconnu.

Le second type d'objet disponible concerne les variables de référence. Celles-ci sont des structures C qui pour un article de la base contiennent:

- sa référence REF
- son type TYPE
- la longueur totale de ses valeurs d'items LENGTH
- ses valeurs d'items

Une référence d'un article est une variable entière qui désigne une zone contenant un pointeur vers l'article lui-même.

Le champ TYPE d'une variable de référence est un entier correspondant au code du type d'article référencé.

Pour déclarer une ou plusieurs variables de référence, le programmeur utilise un type prédéfini dont le nom est R_<name> où <name> est le nom d'un type d'article du schéma.

Les variables de référence constituent donc les arguments des communications entre le programme d'application et les primitives ADL-C.

Le troisième type d'objet mis à la disposition du programmeur concerne les variables d'items d'articles. Ce sont des variables qui contiennent des valeurs d'items correspondant à un type d'article. Pour les utiliser, le programmeur utilise un type prédéfini dont le nom est I_<name> où <name> est le nom d'un type d'article du schéma.

Disposant de ces objets, le programmeur doit disposer de la liste des primitives de manipulation de la base de données.

Signalons tout d'abord que les arguments des fonctions sont soit des variables C classiques, soit des codes des types de chemins et d'articles soit des variables de référence. La plupart des fonctions accèdent à des articles ce qui nécessite la manipulation des variables de référence, en effet tout article obtenu est assigné à une varaible de référence. De plus, après chaque appel à une des fonctions offertes, un code de diagnostic est renvoyé afin de permettre à l'utilisateur de tester la façon dont une opération s'est effectuée.

Liste des primitives ADL-C
---

création du statut de référence:
Donne le statut de "référence" à une variable entière. Celle-ci ,pourra être utilisée pour référencer des articles de la base.

suppression du statut de référence:
Enlève le statut de "référence" à une variable entière

annulation d'une référence:
La variable ne référence plus d'article de la base mais garde son statut

assignation d'une référence à une autre:
Assigne à une variable la référence contenue dans une autre ce qui permet d'avoir deux références accèdant au même article.

test d'égalité de deux références:
Permet de tester si deux variables référencent un même article

tester si une référence est nulle:
Indique si une variable ne référence plus aucun article

ouverture de la base:
Permet l'ouverture d'une base de données dont le nom est fourni

fermeture de la base:
A pour effet de fermer la base de données de nom donné.

accès séquentiel: accès au premier:
Accède au premier article de type spécifié et assigne cet article à la variable de référence.

accès séquentiel: accès au suivant:
Accède à l'article suivant celui qui référencé et l'assigne à cette variable de référence

accès par chemin: accès au premier:
Accès au premier article de type spécifié qui est relié à l'article référencé via le type de chemin, avec assignation à la variable de référence.

accès par chemin: accès au suivant:
Accès à l'article suivant celui qui est référencé et qui est relié à l'article via le chemin donné, avec assignation à la variable de référence.

accès par clé dans un chemin: accès au premier:
Accède au premier article du type spécifié dont la valeur de clé définie dans le chemin se trouve dans la variable donnée. Cet article se trouve dans le chemin qui a pour origine l'article référencé. L'article obtenu est assigné à une variable de référence

accès par clé dans un chemin: accès au suivant:
Accès à l'article suivant celui qui est référencé, qui est du type spécifié et dont la valeur de clé définie dans le chemin se trouve dans une variable déterminée. Cet article se trouve dans le chemin qui a pour origine l'article référencé. L'article obtenu est assigné à une variable de référence.

accès direct:
Accède à l'article de type spécifié et dont on connaît la référence.

création d'article:
Création d'un article de type donné, les valeurs d'items lui sont associées et l'article est relié aux types d'articles souhaités

suppression d'articles:
Suppression d'un article de type donné ainsi que de tous les articles qui lui sont cibles obligatoires et ainsi de suite recursivement.

modification d'article: valeurs d'items:
Modification des valeurs d'items d'un article de type donné et de référence donnée.

modification d'article: valeurs d'ikos:
Modification des valeurs d'items composants d'un identifiant ou d'une clè d'accès d'un article

modification de chemins: attach:
Relier un article de type donné et de référence donnée à un article de référence donnée via un chemin de nom donné

modification de chemins: detach:
Détacher un article, de type donné et de référence donnée.

transférer de la longueur et des valeurs d'items d'une variable de référence vers une autre

ranger les valeurs d'items d'une variable de référence vers une variable d'article

détyper une variable de référence de type R_RECORD.

création d'un texte lié à un objet de la base.

modification d'un texte lié à un objet de la base.

accès à un texte lié à un objet de la base.

2.3.1.3. Interface MAG

Comme cela a été précisé ci-dessus, l'interface MAG est constitué d'un seul point d'entrée. Cette primitive appelée SEM dispose de 9 arguments, ceux-ci permettent de recouvrir toutes les fonctions possibles que l'on a désiré implèmenter. La liste des primitives MAG est donc la même que celle des primitives ADL-C, ce n'est qu'au niveau de la formulation qu'il existe des différences.

2.3.1.4. Enchaînement des primitives

Après avoir examiné la liste des primitives de l'interface ADL-C et celles de l'interface MAG, il est nécessaire de préciser certaines contraintes concernant l'enchaînement des primitives.

En effet, certaines précautions doivent être prises en ce qui concerne l'ordre d'appel des primitives

- l'accès à toute information nécessite la définition préalable de variable de référence
- avant tout traitement sur une base de données, il faut que celle-ci soit ouverte
- la fermeture de la base de données doit toujours s'effectuer avant la clôture du programme
- un accès à un article suivant ne peut se faire que si l'accès au premier est effectué
- la suppression d'un article ne peut avoir lieu que s'il existe dans la base de données
- la modification de valeurs dans un article ne peut se faire que si l'article existe
- relier un article à un autre via un chemin ne peut s'effectuer que si les trois éléments existent
- détacher un article d'un chemin n'est possible que si l'article y était attaché
- supprimer le statut à une variable de référence nécessite l'existence de cette variable de référence
- tester l'égalité entre deux variables de référence nécessite le fait qu'elles ayent toutes les deux le statut
- la modification d'un article nécessite le respect des règles relatives à l'identifiant
- l'ouverture d'une base nécessite qu'elle ne soit pas déjà ouverte

Cette liste non exhaustive de précautions indique simplement que l'utilisateur doit faire preuve de logique lorsqu'il accède aux informations d'une base de données. Quoiqu'il en soit, l'utilisateur pourra toujours vérifier le bon déroulement d'une opération en consultant le code d'erreur. Ce code ne sera peut-être pas toujours précis sur le type d'erreur commis mais sera une précieuse indication pour une correction rapide des erreurs.

code d'erreur
- - - - - - - -
0 : tout s'est bien passé
1 : article non trouvé ou base non trouvée
2 : les contraintes d'identifiant ne seraient plus respectées
10 : la base n'est pas ouverte
11 : la base est déjà ouverte
23 : erreur de code de chemin
24 : erreur de type d'article
27 : erreur de la référence concernée par la primitive
28 : erreur d'une référence autre que celle concernée par la primitive
100: erreur du SGBD (erreur grave devant provoquer l'arrêt de l'exécution)

2.3.2. Aspect interne

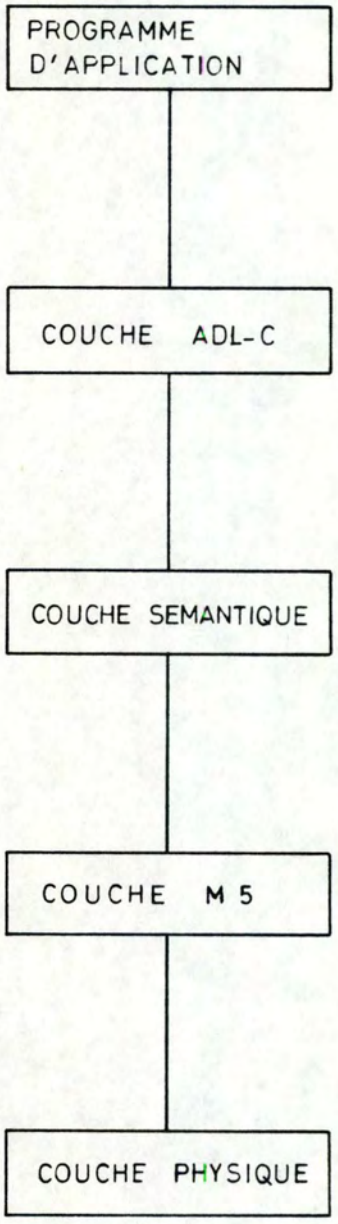
La hiérarchie de l'atelier logiciel orienté bases de données se décompose principalement selon trois niveaux.

Au niveau inférieur, on trouve un module appelé couche physique et dont le but est d'accéder aux données mémorisées dans la base de données. Ces accès sont élémentaires c'est-à-dire qu'ils sont de type: ouvrir un fichier, fermer un fichier, creer un article, spprimer un article, modifier un article, accéder à un article, accéder à l'article suivant d'un fichier.

Au dessus de cette couche physique, on trouve le module M5. Celui-ci assure donc une indépendance vis-à-vis du choix du SGBD de la couche physique. Les concepts du module M5 proviennent du modèle M5 qui sera décrit ci-après.

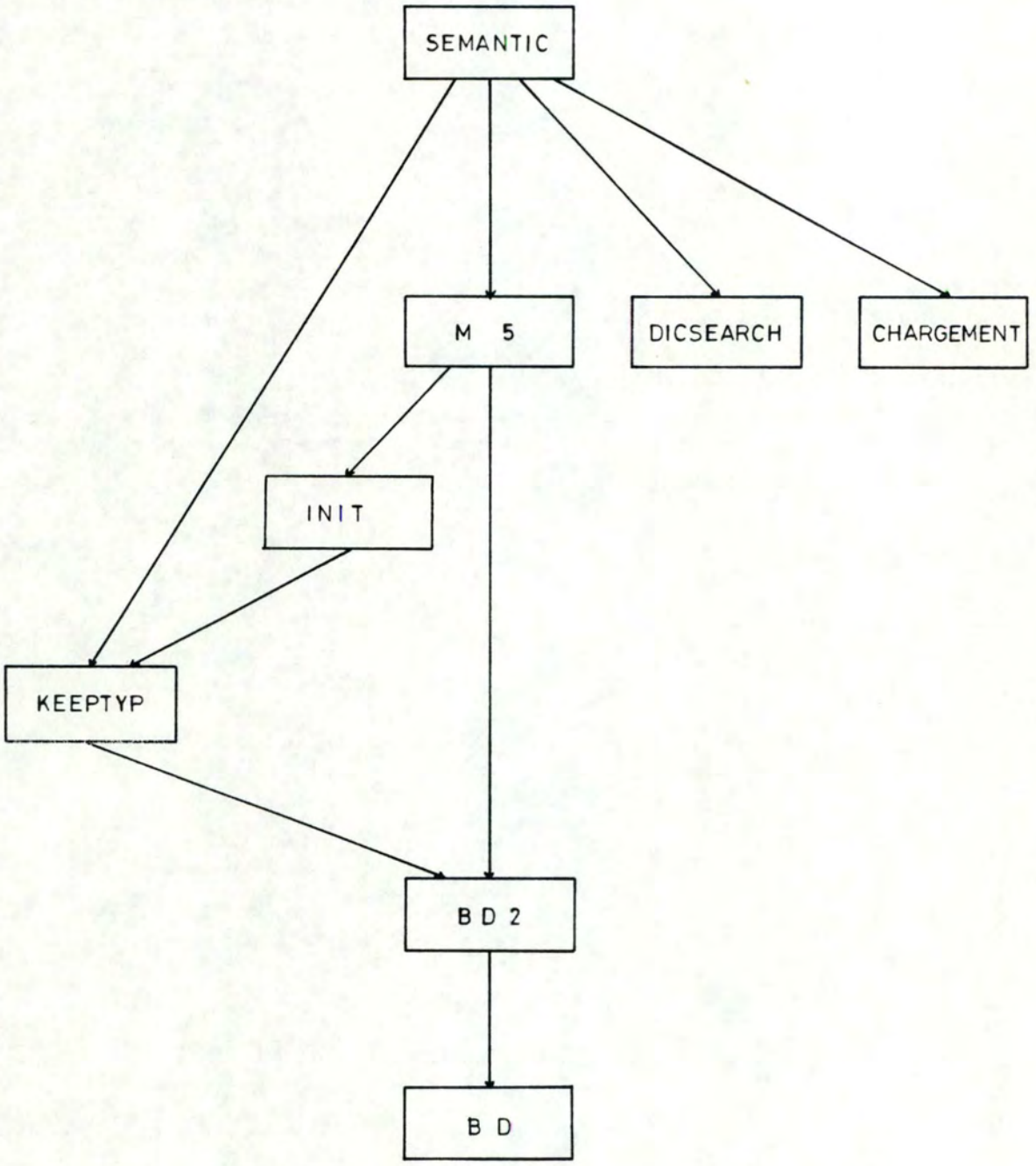
Au dessus du module M5, on dispose de la couche sémantique. Ce module renferme les primitives par lequelles les programmes d'application vont accéder à la base de données: ces primitves constituent l'interface MAG. La compréhension des primitives du niveau sémantique nécessite la connaissance du modèle MAG. De plus étant situé au niveau sémantique, on va avoir besoin des tables. Ces tables appelées tables de conversion vont permettre de traduire un schéma connu sous forme MAG au niveau sémantique en un schéma connu en termes du modèle M5. Cette conversion permet donc le passage du schéma externe connu de l'utilisateur en un schéma interne compréhensible par le SGBD. Le niveau sémantique permet de gérer l'ensemble des informations d'un schéma, on dispose donc grâce à ce niveau à une extension du modèle M5.
Ayant décrit le rôle des tables de conversion (appelées parfois aussi tables internes), on remarquera que toute modification du schéma de données n'entraînera que des modifications au niveau de ces tables mais ne changera en rien les données correspondants à ce schéma.

Au dessus de ces trois couches a été construit une quatrième dont le but est de faciliter l'accès pour le programmeur. Ce module rassemble la liste des primitives ADL-C décritent précédemment. Ces primitives sont plus simples à utiliser et font appel aux primitives du niveau sémantique.



NIVEAUX DE LA HIERARCHIE

L'architecture de l'atelier logiciel se présente comme suit :



DECOUPE EN MODULES

Le module SEMANTIQUE se base sur les tables internes pour interpréter l'accès qui lui est proposé. Cet accès se rapporte au modèle que le module supporte et qu'il doit transformer en un accès au module M5. L'exploitation des tables permet la conversion des données propre au modèle sémantique en données propres au modèle M5.

Le module de CHARGEMENT va lire les tables de conversion pour pouvoir les exploiter.

Le module DICSEARCH réalise des recherches en tables. Il est exploité par le module sémantique dès qu'il y a une conversion à faire entre les données qui sont passées en argument et leurs correspondants dans le modèle M5. Le module de recherche en tables est composé de fonctions de recherche dichotomique qui permettent au départ d'un élément donné d'en fournir la position dans une table ce qui permet au module sémantique d'accéder à cet élément.

Le module M5 réalise les accès au SGBD dont le modèle est le m5 qui sera décrit ci-après. Les différents éléments du modèle m5 sont donc câblés dans le code même du module.

Le module INIT contient un certain nombre de fonctions générales.

Le module KEEPTYP s'occupe de la gestion des références et est exploité par le module sémantique et par le module m5. Au niveau du module sémantique, le module de gestion des références permet de valider les paramètres de références et de tester la compatibilité entre les différents arguments d'un accès. Lorsqu'il est exploité par le module m5, le module ici décrit permet de mettre à jour les clés d'accès pointées par les variables de références. D'autre part il fournit au module m5 le ou les courants des fichiers correspondants aux références fournies.

Le module BD2 agit par rapport au courant d'un fichier. Pour chaque fichier correspond un courant par clé. Ce module met à jour les variables d'état (= les courants) après toute modification du positionnement dans un fichier.

Le module BD permet de traiter les accès élémentaires aux informations.

Le modèle M5.

Le modèle s'appelle M5 parce qu'il comprend 5 élèments de base : OBJECT, OBJECT-0, OBJECT-1, OBJECT-2, TEXT.

Un TEXT est lié obligatoirement à un OBJECT. Tout OBJECT peut avoir 0,1 ou plusieurs TEXT. Un TEXT est défini par un type TXT-TYPE, un groupe TXT-GROUP, une séquence TXT-SEQ, une ligne TXT-LINE.

Un OBJECT est défini par un type TYPE et des données DATA.

Un OBJECT-0 hérite de toutes les propriétés de OBJECT donc de TYPE, DATA et TEXT. Il en est de même pour OBJECT-1 et OBJECT-2. L'héritage des propriétés est défini par la relation "is-a".

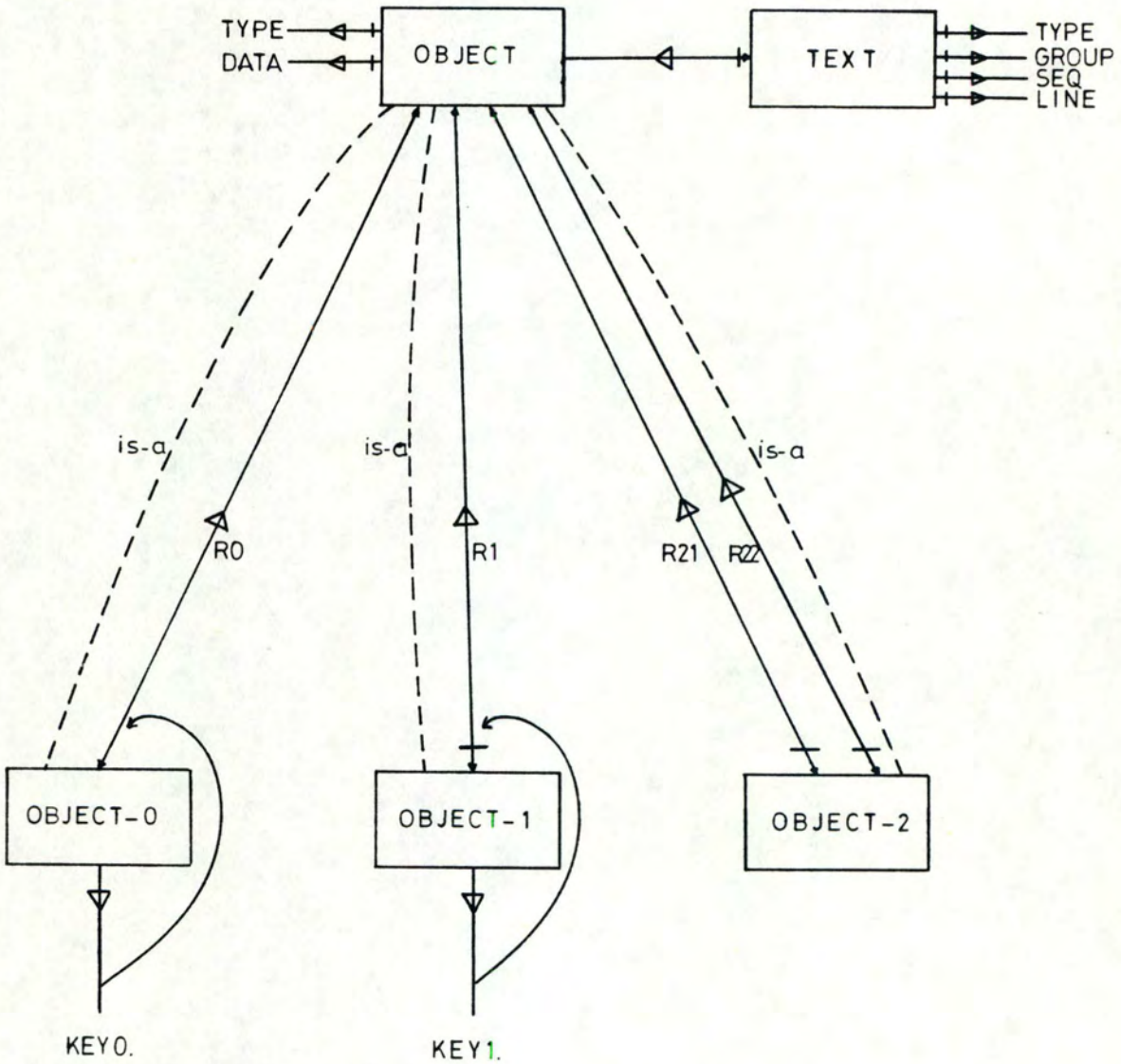
Un OBJECT-0 est un OBJECT lié à un autre OBJECT mais de façon non obligatoire.

Un OBJECT-1 est un OBJECT lié à un OBJECT obligatoirement.

Un OBJECT-2 est un OBJECT lié à deux autres OBJECT obligatoirement.

KEY0 est la clè d'accès dans le type de chemin R0

KEY1 est la clè d'accès dans le type de chemin R1



SCHEMA DU
MODELE M 5

2.4. Gestion des définitions de données

2.4.1. Visions d'une base

Les données d'une base peuvent être réparties selon trois types. Premièrement, on peut distinguer le système d'informations qui est constitué d'un ensemble de données. Deuxièmement, on peut analyser un atelier. Celui-ci permet de donner une structure aux données par l'intermédiaire d'un schéma. Le schéma d'un atelier permet donc de décrire une base de données qui constitue un système d'informations. Troisièmement, on dispose du méta-atelier. Le méta-schéma associé au méta-atelier permet de décrire n'importe quel atelier et donc n'importe quel système d'informations.

Ce qui est remarquable dans le méta-schéma décrit ci-après, c'est qu'il est unique peu importe qu'on se place au niveau d'un atelier ou du méta-atelier. En effet, ce méta-schéma permet de décrire n'importe quel atelier mais peut aussir servir à décrire le méta-schéma lui-même; le méta-atelier est un atelier parmi les autres et peut donc être décrit par le méta-schéma, sa particularité est qu'il permet de décrire d'autres schémas.

Nous allons maintenant décrire les différents éléments du méta-schéma en adoptant la représentation MAG (modèle d'accès généralisé). Ensuite nous expliquerons un exemple qui illustre les différentes visions décrites ci-dessus.

Description sous forme MAG du méta-schéma de l'atelier

Le méta-schéma de l'atelier est un sous-ensemble du schéma conceptuel général décrit dans (HAI-86a). La description ci-après reprend les définitions nécessaires à la compréhension du méta-schéma de l'atelier constituant du méta-atelier.

Puisque le méta-schéma est introduit dans la base de données, il doit respecter les contraintes de conformité du SGBD réel. Ces contraintes ne doivent pas être connues à ce stade mais elles justifient certains choix dans la description du schéma.

L'élément principal du méta-schéma est le SYSTEM qui représente un système d'informations. Chacun d'eux est identifié par un nom SYS-NAME, un état STATUS ainsi que par les DATE1 et DATE2 indiquant respectivement le début de construction du système et la dernière modification.

Toute description de données se présente sous forme d'un SCHEMA relié au SYSTEM. Un SCHEMA est identifié par son nom SCH-NAME. Il est caractérisé par sa CLASS qui indique que le schéma est conceptuel ou que l'on traite de la conception physique ou encore de la conception logique. Le TYPE indique si on décrit le schéma conforme, le schéma des accès possibles ou les schéma des accès nécessaires. Ces différents termes sont repris de la démarche de conception de base de données (HAI-86).

L'élément de base d'un schéma est l'ENTITY-T qui représente les notions traditionnelles de type d'entité, schéma de relation, relation, type d'article, segment,etc, et que l'on retrouve dans toute description structurée de données. L'E-NAME identifie l'entité par son nom. Le SHORT-NAME permet de fournir un nom plus court. L'E-CODE est un code identifiant l'entité autrement que par son nom. La POPULATION informe du nombre d'entités de ce type. Les LOG-LENGTH et PH-LENGTH représentent respectivement la longueur logique et physique de l'entité.

Un type d'association entre types d'entités est représenté par un LINK. Le L-NAME identifie le link par son nom. Le SHORT-NAME est un nom plus court pour identifier le link. Le L-CODE identifie le link. Le DEGREE indique le nombre de liens représenté par un link. Dans ce cas présent, un link sera toujours de degré 2. Le schéma peut accéder directement à tous les link et inversément.

Un ROLE représente la participation d'un type d'entité à un link. Le nombre de rôle indique le degré du link. Le R-NAME et le R-CODE identifient chacun, respectivement par un nom et un code, le rôle. Le SHORT-NAME et le SYNONYM permettent de nommer autrement le rôle. Les caractéristiques MIN-CON et MAX-CON représentent les connectivités minimale et maximale. L'attribut PATH-NAME donne le nom du chemin pour le rôle. Ce nom n'existe pas toujours puisqu'il peut arriver que l'on refuse l'accès d'une entité vers une autre. Deux rôles existent toujours lorsqu'il y a un link qui est déclaré mais les accès peuvent être refusés entre les deux entités. L'attribut MULTIPLE indique si la participation au rôle est composée d'une seule ou de plusieurs entités.

Un ENT-ROL identifie la participation d'une entité à un rôle permettant de définir une participation multiple éventuelle. En effet, si plusieurs entités participent à un rôle, il existe plusieurs ent-rol liés au rôle.

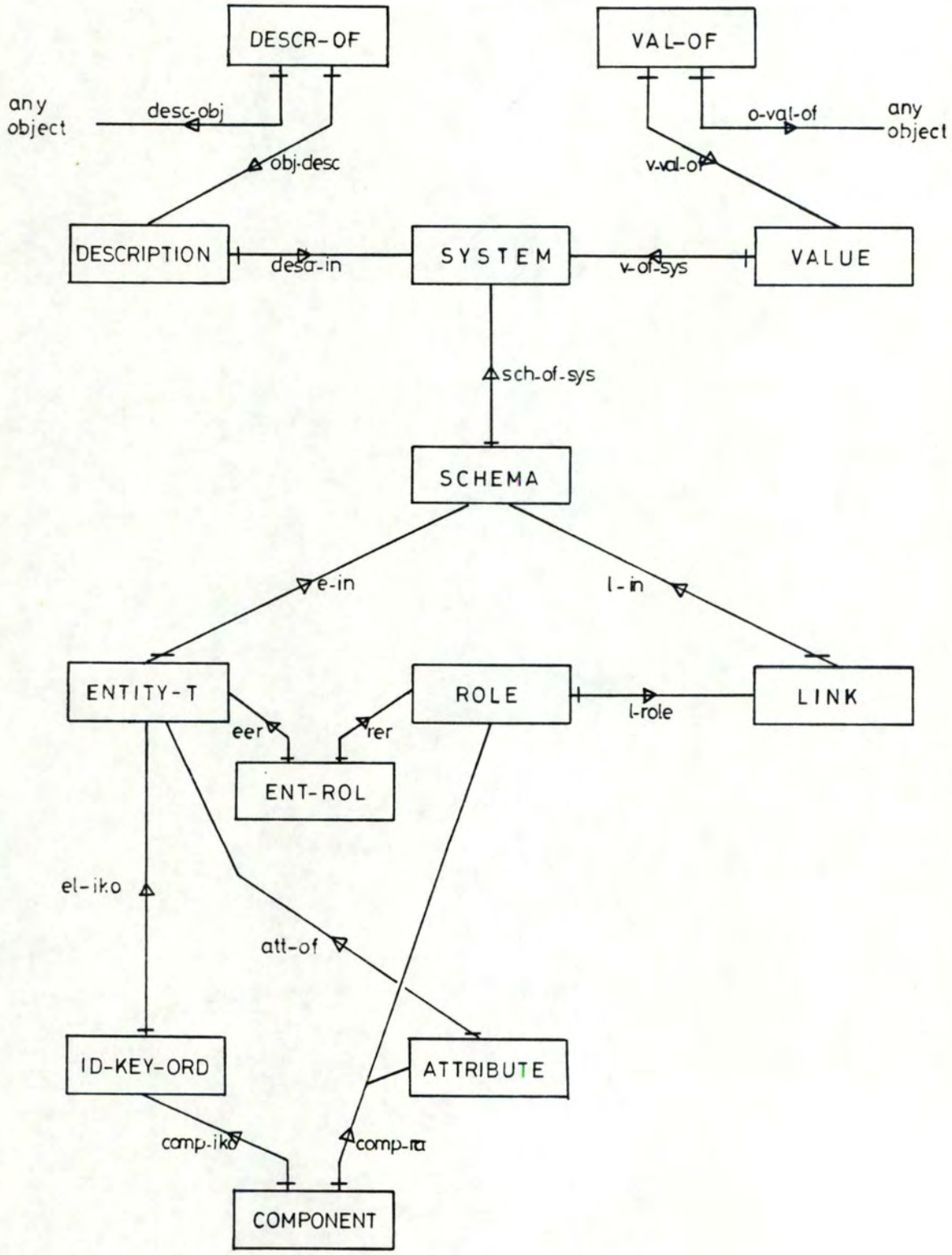
Toute propriété caractérisant une entité est représentée par un ATTRIBUTE dont l'identifiant est par le nom AT-NAME et par le code AT-CODE. Le SHORT-NAME et le SYNONYM permettent de renommer l'attribut. Le TYPE indique un des choix suivants: virtuel, réel, calculé, temporaire, banal, de contrôle,... Le STATUS informe de l'état. La DATE indique la dernière mise-à-jour tandis que le LENGTH donne la longueur maximum de l'attribut. Le MIN-REP et le MAX-REP indique la répétitivité minimale et maximale. Le DECIM donne le nombre de décimale. Le NUM-ORD donne la position de l'attribut par rapport aux autres attributs de l'entité. Le FORMAT indique le format des valeurs de l'attribut, celui-ci est "T" pour informer que l'attribut est de type texte. Un attribut de type texte permet de décrire sous forme d'un texte libre une entité. Plusieurs attributs de type texte peuvent être associés à une entité.

Pour chaque entité on peut spécifier un ou plusieurs ID-KEY-ORD. Un id-key-ord,noté parfois iko, représente les identifiants, les clé d'accès et l'ordre dans lequel sont rangés les élèments d'une séquence et donc l'ordre dans lequel on y accède. Le type est "i" pour indiquer un identifiant et "k" pour une clé identifiante.

Un COMPONENT est le composant d'un iko. Un iko a deux composants: soit deux rôles, soit un rôle et un attribut. En effet, un iko peut être représenté par deux rôles qui sont les deux rôles auquels participe l'entité. L'autre représentation d'un iko se fait par un attribut qui est identifiant dans un chemin. Tout composant est identifié par une occurrence de l'entité des ikos et par une occurrence soit du rôle soit de l'attribut. Le TYPE indique si c'est un rôle ou un attribut. Le SEQ-NBR informe de l'ordre des composants de la clé de tri et DIRECT du caractère ascendant ou descendant.

A tout objet du schéma on peut associer via VAL-OF un ensemble de valeurs. Une VALUE est représentée par un nom V-NAME, un TYPE, un FORMAT, une EXPRESSION ainsi que par MEANING donnant la signification de la valeur. ROLE caractérise le rôle joué par la valeur, SEQ-NBR un numéro de séquence pour ordonner des suites de valeur et PROBABILITY la probabilité d'apparition d'une certaine valeur.

Tout type d'objet peut faire l'objet d'une DESCRIPTION caractérisée par une CLASS, un TYPE, un état donné par STATUS une DATE de dernière mise-à-jour et le corps de la description donnée par le TEXTE. Une description est associée à l'objet qu'elle décrit via DESCR-OF.


DESCR-OF
VAL-OF
any object
desc-obj
obj-desc
o-val-of
any object
v-val-of
DESCRIPTION
desc-in
SYSTEM
v-of-sys
VALUE
sch-of-sys
SCHEMA
e-in
l-in
ENTITY-T
ROLE
LINK
l-role
eer
rer
ENT-ROL
el-iko
att-of
ID-KEY-ORD
ATTRIBUTE
comp-iko
comp-ra
COMPONENT

META - SCHEMA

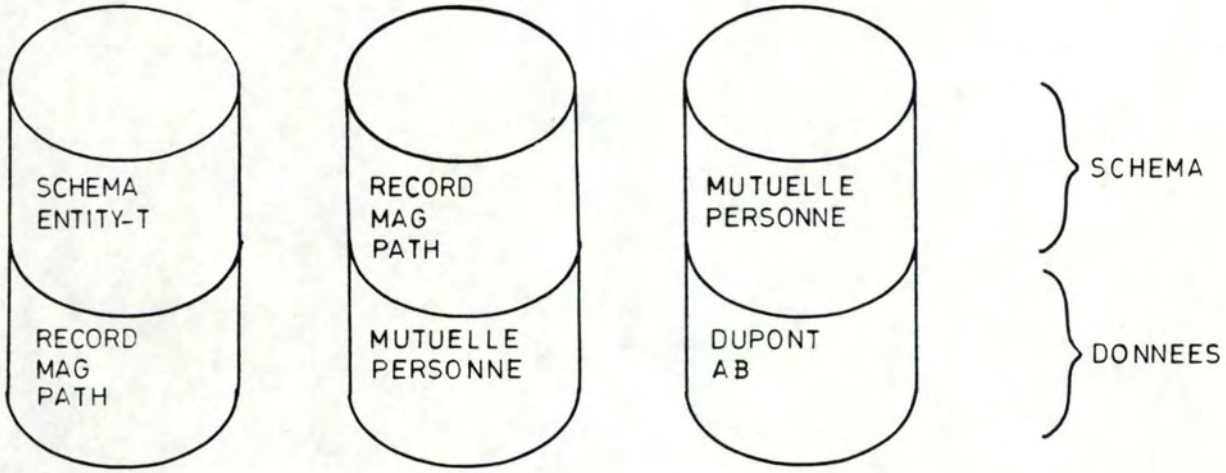
exemple des diférentes visions des données:

Par l'intermédiaire du méta-schéma, on peut décrire différents schémas.

a) un SCHEMA de nom MAG
   un ENTITY-T de nom RECORD
   un ENTITY-T de nom PATH

b) un RECORD de nom MUTUELLE
   un RECORD de nom PERSONNE

c) une PERSONNE de nom DUPONT
   une MUTUELLE de nom AB



2.4.2. Formes de schémas

Comme cela a été exprimé dans la première partie, un schéma de données peut se présenter sous plusieurs formes.

D'un côté, on peut voir la forme enregistrée d'un schéma. Celle-ci correspond au schéma du dictionnaire de données.

D'un autre côté, on dispose du schéma source qui est le sous-ensemble défini d'un système d'informations. Ce schéma constitue l'entrée du compilateur de schéma.

Troisièmement, on distingue la forme compilée d'un schéma qui se compose d'une part du schéma externe et d'autre part du schéma interne.
Lorsque l'utilisateur a décrit le schéma source de la base de données, le compilateur de schémas permet de créer les versions compilées du schéma.

D'une part on dispose du schéma des données qui sera décrit sous forme de 4 tables appelées tables du schéma: une table reprenant certaines informations sur les entités, une pour tous les attributs des entités, une autre pour l'ensemble des rôles du schéma et enfin une table pour indiquer les entités origines des rôles du schéma. Ces tables externes seront exploitées par le définisseur d'écran, par l'interpréteur de commandes, par l'analyseur syntaxique et vraisemblablement par d'autres outils.

D'autre part, on possède le schéma interne de la base de données. Au départ du schéma externe, un certain nombre de tables sont créées. Ces tables appelées tables internes, permettent la conversion du schéma externe en un schéma interne compréhensible par le SGBD. 4 tables sont nécessaires à cette description: une table reprenant des informations sur les objets (Modèle M5) c'est-à-dire sur les entités du schéma, une table pour les relations existant dans le schéma (objets 2 du modèle M5), une table des codes reprenant les codes des origines des chemins ainsi que les codes des relations pour lesquelles les entités sont cibles et enfin une quatrième table reprenant des informations sur les iKos (identifier-key-order).

2.4.3. Conformité d'un schéma

Lorsqu'un utilisateur introduit un schéma de données par l'intermédiaire du méta-schéma, il décrit un ensemble d'èlèments mais qui peuvent ne pas être conformes au SGBD. En effet, toutes les possibilités su schéma MAG ne peuvent être acceptées dans l'atelier.

Le rôle d'un vérificateur de conformité est de rendre un schéma conforme au SGBD. Cette conformité se réalise par vérification d'un certain nombre de règles. Deux options sont possibles par rapport à l'analyse de la conformité. On peut soit construire un vérificateur paramétrable ou non paramétrable. On trouvera dans (CHAR-MUL-85) une analyse sur les avantages et les inconvénients de ces deux options.

liste des contraintes pour l'atelier logiciel:

- E_NAME est identifiant d'un ENTITY-T dans un schéma

- AT_NAME doit être un identifiant d'un ATTRIBUTE dans un ENTITY-T

- la connectivité des types de chemins est toujours "one-to-many"

ceci implique que :

si la connectivité maximale du rôle est "99999" alors la connectivité minimale du rôle est "0" et la connectivité maximale du rôle inverse est "1".

si la connectivité maximale du rôle est 1 alors la connectivité maximale et rôle inverse est "99999" et la connectivité minimale du rôle inverse est "0".

- un type d'article ne peut être cible obligatoire que de deux types de chemins maximum

- pour définir un texte relatif à une entité, on définira un attribut dont le format sera "T"

- en dehors des types de chemin ayant pour cible un type d'article "texte" il n'existe pas de clè d'accès ni de clè de tri

- un item d'un type d'article peut être identifiant et/ou clè dans tous les types de chemin où ce type d'article est obligatoire

Un iko est toujours défini dans un type de chemin.

- Si un type d'article possède un identifiant dans un type de chemin non obligatoire, alors ce type d'article possède un seul identifiant et n'est cible que de types de chemins facultatifs

- seuls deux types de chemins obligatoires peuvent former un groupe identifiant. Cette contrainte est la seule pouvant porter sur le type d'article cible de ces types de chemins

- un attribut identifiant ne peut être de type "texte"

- le type d'un composant d'un identifiant est "role" ou "atr"

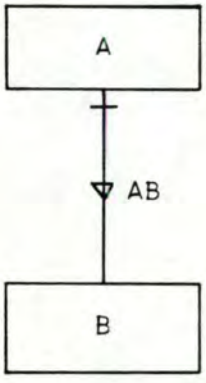
- il ne peut exister qu'un seul iko dans un chemin

2.4.4. Versions d'un schéma

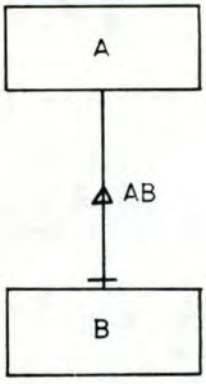
Le but de ce paragraphe est d'analyser dans quelle mesure le SGBD peut accepter des modifications d'un schéma de données sans qu'il y ait des répercutions sur les données et les programmes.

Lorsqu'on traite de versions d'un schéma, on désire analyser l'hypothèse d'une transformation d'un schéma. Beaucoup de prudence est nécessaire car toute information ne peut être acceptée si l'on veut conserver l'intégrité de la base de données.

Prenons par exemple le cas suivant:



Il est impossible d'accepter d'ajouter le type d'article B et le type de chemin AB sinon on doit vérifier que tout article B doit être lié à un article A qui doit exister. Par contre dans le schéma ci-après, il est tout à fait acceptable d'ajouter le type d'article B.



Nous analysons ci-après les conséquences qu'entraîne la suppression ou l'ajout d'un article d'un type d'article du méta-schéma. Nous ne décrivons pas les problèmes de transformation de schéma, ceux-ci sont traités dans le document (CHAR-MUL 85).

Pour ce qui concerne la suppression et l'ajout nous avons traité tous les cas sans restriction. Lors de l'implémentation du traitement des versions, il est certain que diverses restrictions devront être prises en vue de faciliter la gestion. L'acceptation de tous les cas de figures entraînerait une gestion trop importante pour la conservation de l'intégrité de la base de données.

Suppression

Hypothèse: la suppression suppose l'existence de ce que l'on désire supprimer

- la suppression d'un SCHEMA entraîne des suppressions pour des ENTITY-Ts, des ROLEs, des LINKs, des IKOs, des ATTRIBUTEs et des COMPONENTs

En effet, si on désire supprimer un SCHEMA, tous les éléments le constituant seront supprimés

- la suppression d'un LINK entraîne des suppressions pour des ROLEs, des COMPONENTs et des IKOs.

En effet, si on supprime un LINK il faut: supprimer les ROLEs du LINK, supprimer les IKOs dans le chemin, l'ENTITY_T côté many de la relation est cible obligatoire d'un chemin en moins, si un des ROLEs était COMPONENT d'un IKO, l'autre partie de l'IKO devient un chemin obligatoire mais non identifiant, un ou plusieurs ENTITY_T sont origines d'un chemin en moins

- la suppression d'un ROLE entraîne des suppressions de ROLEs, de COMPONENTs et d'IKOs

La suppression ne sera acceptée que si la connectivité maximale du ROLE est "99999" (équivaut au N de la notion de classe fonctionnelle dans le modèle MAG) et cela uniquement dans le cas de chemin multi-origine. Alors seulement l'ENTITY_T sera origine d'une chemin en moins que précédemment.

- la suppression d'un ENTITY-T entraîne des modifications pour les ROLEs, les LINKs, les ATTRIBUTEs, les IKOs, les COMPONENTs.

En effet, il est nécessaire alors de supprimer les ATTRIBUTES de l'ENTITY_T, supprimer les IKOS de l'ENTITY_T, supprimer les ROLEs joués par l'ENTITY_T, supprimer les LINKs entre cette ENTITY_T et les autres, supprimer les ATTRIBUTES qui sont IKOs dans un chemin entre cette ENTITY_T supprimée et une autre, si un ROLE cible entre l'ENTITY_T supprimée et une autre ENTITY_T était COMPONENT d'un identifiant, l'autre COMPONENT d'un identifiant devient chemin obligatoire mais non identifiant, si l'ENTITY_T est one-to-many vers une autre alors l'autre ENTITY_T est cible d'un chemin en moins sinon l'ENTITY_T est origine d'un chemin en moins.

- la suppression d'un ATTRIBUTE entraîne des modifications sur les COMPONENTs et les IKOs.

En effet, si l'ATTRIBUTE est IKO il faut supprimer l'ATTRIBUTE et les COMPONENTs de cet IKO et l'IKO lui-même sinon il suffit de supprimer l'ATTRIBUT

- la suppression d'un IKO entraîne des modifications de COMPONENT.

En effet, si l'IKO était composé de deux ROLEs, il faut supprimer la participation de ces deux rôles comme composants d'un IKO. Si l'IKO est composé d'un ATTRIBUTE et d'un ROLE il faut supprimer ces deux COMPONENTs et cet IKO.

- la suppression d'un COMPONENT est impossible, on ne peut supprimer que deux COMPONENTs à la fois mais alors cela revient à supprimer un IKO.

Ajout

- l'ajout d'un SCHEMA suppose l'existence d'un SYSTEM

- l'ajout d'un LINK suppose la liaison avec un SCHEMA et la définition de deux ROLEs.

- l'ajout d'un ROLE suppose l'existence d'un LINK et des ENTITY-T correspondants

Le ROLE sera attaché à un LINK sauf si celui-ci possédait déjà deux ROLEs. Si le ROLE côté one de la relation existait déjà, il faut que le ROLE que l'on ajoute soit un ROLE côté many de la relation et inversément.

Si on désire ajouter une ENTITY_T comme ROLE côté many d'une relation existante, on refuse cet ajout. Si l'ajout se fait du côté one d'une relation existante, le chemin devient multi-origine (ou reste multi-origine s'il l'était déjà) et l'ENTITY_T concernée devient origine d'un chemin en plus. L'ENTITY_T devient cible supplémentaire pour l'autre ENTITY_T participant au ROLE.

Si on ajoute une ENTITY_T que l'on associe à un nouveau LINK alors: si c'est le côté one de la relation, l'ENTITY_T est origine d'un chemin en plus. Si c'est du côté many, si le ROLE est non obligatoire, l'ENTITY_T devient cible d'un chemin en plus, si le ROLE est obligatoire alors si l'ENTITY_T est déjà cible de deux chemins obligatoires, on refuse l'ajout sinon on accepte l'ajout et l'ENTITY_T cible est cible d'un chemin en plus.

- l'ajout d'un ENTITY-T nécessite l'existence d'un SCHEMA

Il faut voir si l'ENTITY_T n'existe pas déjà.

- l'ajout d'un IKO nécessite l'existence de l'ENTITY-T correspondant

On ajoute donc deux COMPONENTs qui sont soit deux ROLEs, soit un ROLE et un ATTRIBUTE.

- l'ajout d'un ATTRIBUTE demande que l'ENTITY_T existe.

Le nom de l'ATTRIBUTE n'existe pas encore.

- l'ajout d'un IKO n'existe que si l'on ajoute directement un deuxième COMPONENT qui formeront l'IKO. Un COMPONENT au maximum est du type ATTRIBUTE. Si un des COMPONENTs est ATTRIBUTE, celui-ci ne peut pas déjà participer à un autre identifiant. Si un des COMPONENTs est ATTRIBUTE, il faut que celui-ci existe dans la liste des ATTRIBUTEs de l'ENTITY_T. Si c'est un ROLE, il faut que ceLui-ci ne participe pas déjà à un autre identifiant.

2.4.5. Environnement programmeur

Lorsque la définition du schéma est effectuée, il est possible de fournir au programmeur un environnement correspondant à ce schéma.

C'est ainsi que l'on pourra fournir une définition par entité et une par nom de type de chemin. Cette définition consiste à associer chaque nom de type d'article ou du type de chemin à son code défini par le programmeur.

D'autre part, par la fonction "typedef" du langage C, il est possible de disposer d'un nouveau type de données par entité définie. Ce nouveau type de données reprend une structure comprenant tous les attributs de l'entité.

Enfin, pour chaque entité, il est généré une fonction permettant, lors de l'exploitation du dictionnaire de données, de créer une entité de nom donné. Cette fonction de création fait partie de l'ensemble des primitives décrites dans l'interface entre la base de données et les programmes utilisateurs.

Chapitre 3 : Réalisation

3.1. Transport de l'interpréteur de commandes

3.1.1. Architecture

Nous avons vu précédemment que l'architecture principale de l'atelier logiciel orienté bases de données se décompose en trois niveaux; tout d'abord la couche sémantique, ensuite la couche M5 et enfin la couche physique.

Lors de la première réalisation de l'atelier, l'implémentation a eu lieu sur un mini-ordinateur VAX. Ayant eu à ce moment-là des difficultés avec le langage VAX/C, il a été nécessaire de remplacer la couche physique prévue en langage C par des appels à des fichiers RMS/COBOL.

Tout le logiciel a donc été testé par manipulation de fichiers COBOL avec les avantages et les inconvénients que cela comporte.

3.1.2. Objectif

Pour l'implémentation initiale de l'atelier, ce sont les langages C et Cobol qui ont donc été utilisés.

Il a ensuite été décidé de transporter cet atelier sur micro-ordinateur. Le choix s'est porté sur le langage Lattice C associé à un dbase 3 (DBC de Lattice) pour remplacer les fichiers RMS/Cobol. Le but principal de ce transport est d'assurer une portabilité du produit. L'accès au micro-ordinateur pour cet atelier permet de pouvoir lors de l'exploitation utiliser des outils propres aux micro-ordinateurs qui facilitent la manipulation de l'atelier.

La première étape du travail a été de traduire en Lattice C les programmes initialement construits en Cobol en y intégrant l'utilisation du Dbase 3. Pour se faire nous avons consulter (LATT-82), (CLAR-81) et (LATT-85).

Deuxièmement, il a été nécessaire de transformer les programmes C initiaux pour les rendre conformes au Lattice C tout en essayant de trouver des solutions égales pour le Vax/C et le Lattice C.

Par la suite, les programmes C du micro-ordinateur ont été transférés sur le mini-ordinateur ce qui a permis d'assurer une cohérence parfaite entre les deux systèmes excepté bien sûr la manipulation élémentaire des fichiers. La seule différence qui persiste entre les deux systèmes est donc bien que sur le mini-ordinateur on utilise des fichiers Cobol et sur le micro-ordinateur un Dbase 3.

Le but de ce chapître n'est pas de montrer en détails les difficultés rencontrées lors du transport du logiciel mais de tirer parti de cette expérience pour fournir une liste de précautions nécessaires en vue de réaliser la meilleure portabilité de programmes.

3.1.3. Problèmes rencontrés

a) discordance du séquentiel indexé par rapport au DBC

Afin de réaliser le transport des programmes du mini-ordinateur vers le micro-ordinateur, la transformation des programmes a du être envisagée.

La modification des instructions Cobol en instructions DBC (Dbase 3 de Lattice) n'est pas du tout automatique.

Premièrement, il faut disposer de la spécification précise du module accédant aux fichiers Cobol; cette spécification permettant de connaître exactement le rôle des différentes primitives.

Deuxièmement, on doit connaître les spécifications exactes quant à la manipulation des fichiers Cobol par rapport à la manipulation des fichiers DBC. En effet, l'organisation des fichiers séquentiels indexés en Cobol ne correspond pas du tout à celle du DBC.

En Cobol, dans un fichier séquentiel indexé, on dispose d'un seul courant par fichier quel que soit le nombre d'index qui sont déclarés. Rappelons qu'un courant indique toujours le dernier article manipulé dans une séquence déterminée. Ceci indique que la gestion du courant est simple puisqu'il n'y en à qu'un seul par fichier et qu'il est donc facilement contrôlable; si on accède séquentiellemnt ou par clé à un fichier Cobol, ce sera toujours le même courant qui sera mis à jour.

En DBC, chaque base de données consiste en un ensemble de fichiers liés entre eux. Les données proprement dites sont stockées dans des fichiers dont l'extension est DBF. Les informations sur les index sont contenues dans des fichiers d'extension NDX. Chaque enregistrement d'un index contient une clé et le numéro de l'enregistrement dans le fichier d'extension DBF. A chaque fichier d'index NDX correspond un courant, de même il existe un courant par fichier DBF. Lors de la programmation, il faut donc tenir compte de tous ces courants. Lors par exemple de la création d'un enregistrement, il est nécessaire de créer une entrée dans le fichier DBF mais également une entrée dans chaque fichier NDX. Le DBC offre une plus grande souplesse puisque chaque index peut être géré séparément . Bien sûr cette facilité complique nettement la gestion des index et des courants des fichiers.

b) Restrictions de programmation.

Le transport de l'atelier a montré que la programmation sur micro-ordinateur est plus restrictive. Si l'on veut assurer une portabilité maximale des programmes, il est nécessaire de prendre quelques précautions:

Par exemples:

- Un compilateur peut ne prendre en compte que les 8 premiers caractères d'une chaîne de caractères pour différencier les identifiants des variables alors qu'un autre peut accepter de différencier des chaînes plus longues. Une longueur de 8 caractères semble être la norme minimale la plus courante au niveau de la différenciation des identifiants.

- L'alignement des zones mémoires peut être différent d'un ordinateur à l'autre et causer des problèmes d'accès à certaines données. En effet, si j'ai par exemple une structure formée de 3 caractères suivie d'une chaîne de 4 caractères, dans un cas on dispose d'une zone équivalente à 7 caractères alors que dans un autre la place mémoire réservée est de 8 caractères. Ces 8 caractères se justifient par le fait que la chaîne de 4 caractères est alignée sur une frontière de mots. La place réservée est donc de 4 caractères au lieu de 3 caractères. Dans ce cas, un caractère inaccessible aura été introduit entre la structure et la chaîne de caractères. Ce même problème se pose lorsqu'on définit une chaîne de caractères de longueur impaire suivie d'une zone définissant par exemple un entier.

- Les types de données ne sont pas définis de la même façon d'un ordinateur à l'autre. Un "long integer" peut valoir 32 bits pour l'un et 64 bits pour l'autre. L'utilisation de différents types peut donc causer des problèmes lors par exemple du passage des paramètres d'une fonction. Si les longueurs par défaut ne sont pas les mêmes, les résultats peuvent être incorrects. La portabilité des programmes implique donc l'utilisation du plus grand nombre possible de "standards" c'est-à-dire que lorsque plusieurs solutions sont possibles, il faut choisir celle qui est valable dans le plus grand nombre de cas.

c) Découpe modulaire et fonctionnelle

La qualité d'une découpe est très importante. Celle-ci sera particulièrement marquante lorsque, transportant un ensemble de modules de programmes, il est nécessaire de réécrire un de ceux-ci.

Dans (AVL-84) on peut distinguer plusieurs intérêts marquants d'une bonne hiérarchie c'est-à-dire d'une organisation intelligente de différents composants.

Lorsqu'il faut définir les différents composants de cette hiérarchie, que nous appelons aussi modules de la hiérarchie, il est important de retenir trois qualités primordiales: la forte capacité de cacher de l'information, le faible degré de couplage et la forte cohésion interne.

d) Tests

Lorsque des programmes sont écrits, il est toujours nécessaire d'effectuer des tests rigoureux. Un test sommaire est inutile car il n'implique pratiquement aucune garantie sur le respect des spécifications par les programmes.

Mais, il ne suffit pas de tester les programmes de façon intuitive, des jeux de tests rigoureux doivent être spécifiés; on y fournira les données introduites ainsi que les résultats à obtenir.

Pourquoi est-ce important? Parce que lorsqu'on effectue le transport des programmes d'un ordinateur vers un autre, les mêmes tests peuvent resservir. De plus, si des programmes spécifiques de tests sont écrits, ils seront inutilisables lors du transport des programmes s'ils ne sont pas spécifiés. Si aucun test rigoureux n'est effectué, ce sont les programmes appellant qui en subiront les conséquences. Ces programmes seront difficiles à mettre au point puisque lorsqu'une erreur apparaîtra il sera parfois impossible de savoir si l'erreur provient du programme testé ou bien des programmes appelés insuffisamment testés.

Au niveau de l'atelier logiciel, il existe un programme de test interactif pour chaque niveau de la hiérarchie. De plus, un programme non interactif teste sur un exemple toutes les possibilités de combinaisons des primitives MAG. Ces différents programmes de tests ont été testés sur le mini-ordinateur mais ont pu être utilisés sur le micro-ordinateur.

e) Spécifications

Lorsqu'on analyse une hiérarchie de modules, on pourrait se dire que toute personne qui utilisera cette hiérarchie n'aura jamais besoin que des primitives d'accès. Il est vrai qu'un programmeur construisant des programmes d'application à un niveau supérieur n'a aucunement besoin de connaître la structure et la spécification de cette hiérarchie. On pourrait donc avoir tendance à négliger ces spécifications internes à la hiérarchie. Mais, ce serait oublier l'élément capital en programmation qui est que tout programme écrit est amené à être modifié. De plus, ces modifications sont rarement effectuées par la ou les personnes qui ont écrits les programmes initiaux. C'est dire combien une spécification des modules et des programmes est importante. Pour être certain qu'un module fasse ce qu'on veut, il faut qu'on dise ce qu'on veut qu'il fasse.

f) Versions utilisées

Lorsqu'on désire transporter des programmes d'un ordinateur vers un autre, il est nécessaire d'assurer que l'on fournit bien la dernière version des programmes et que celle-ci est bien identifiée. Dans le cas contraire, on serait amené à chercher des solutions à des erreurs déjà corrigées dans une version postérieure à celle dont on dispose. La perte de temps peut être considérable si ce genre d'erreur se multiple. D'où l'importance de tenir à jour une liste des différentes versions des programmes et des modifications apportées. A tout moment, on doit savoir où on en est entre les différentes versions. De plus, cela permet de toujours redémarrer un système avec une ancienne version s'il se produit un incident.

En ce qui concerne le transport proprement dit. Si après avoir commencé le transport sur un autre ordinateur, la version implémentée sur l'ordinateur initial subit des modifications, il est impératif de les notifier pour pouvoir les répercuter facilement. Sinon, on constaterait par la suite des différences entre les versions ce qui pourrait amener des recherches importantes.

g) Manipulation des outils

Nous présentons ici deux remarques générales qui concernent la programmation en C sur un micro-ordinateur.

La première remarque concerne le micro-ordinateur lui-même. Lors du travail effectué sur le micro-ordinateur, il peut se produire différentes pannes ou mauvais traitements. En effet, si certains ordinateurs sont laissés à la disponibilité d'un grand nombre de personnes, toutes les manipulations ne peuvent être contrôlées. Ceci peut amener des dommages importants vu l'incompétence de certaines personnes à manipuler les ordinateurs. D'autre part, il peut arriver que le système lui-même rencontre un certain nombre de problèmes. Dans ce cas, il se peut que divers fichiers deviennent inaccessibles. Ces deux cas montrent, si c'était nécessaire, à quel point la notion de backup est importante même sur un micro-ordinateur que l'on croit maîtriser plus facilement. De plus, une procédure de redémarrage doit toujours être envisagée à tout moment de l'évolution d'un projet pour éviter de perdre trop de temps lors d'incidents.

La deuxième remarque est relative à la programmation en Langage C. En effet, l'utilisation de pointeurs peut entraîner des difficultés incontrôlables. Il peut arriver par exemple qu'une mauvaise manipulation des pointeurs bloque complètement un système ce qui nécessite le redémarrage complet de l'ordinateur. Une autre erreur plus complexe à détecter est le fait que la manipulation de pointeurs peut fournir des résultats non conformes à ceux qui étaient prévus mais ceci pas nécessairement lors de la première exécution du programme. Il se peut par exemple qu'un programme fournisse des résultats corrects pour un nombre limité de données alors que si l'on augmente le nombre d'itérations, les résultats deviennent incorrects. La conclusion de tout ceci est que la manipulation de pointeurs en langage C est très délicate et que toute utilisation doit être bien contrôlée par le programmeur car aucune information ne sera fournie à l'utilisateur lors de la compilation ou lors de l'édition des liens (link).

3.1.4. Spécifications du module de la couche physique

Après avoir analysé les différences au niveau de la gestion entre les fichiers Cobol et le Dbc et après avoir réuni les informations permettant de connaître les conditions d'utilisation de ce module de la couche physique nous avons pu déduire les spécifications suivantes :

objectif :

L'objectif de la couche physique est de réaliser l'ensemble des traitements élémentaires vis-à-vis des fichiers. Le module réalise sept opérations différentes : ouverture , fermeture d'une base de données, création, mise-à-jour, suppression,lecture d'un record, lecture du record suivant.

pré-conditions :

1. Toutes les clés des enregistrements sont déclarées caractères

2. On suppose que le nom de la base de données, passé en argument pour l'ouverture et la fermeture, est correct; les contrôles étant effectués par les modules de niveaux supérieurs.

3. Lors de l'exploitation sur micro-ordinateur, l'ouverture des fichiers de la base de données n'entraînera pas l'ouverture simultanée de plus de 15 fichiers. En effet, le Dbc (dbase 3 de Lattice) ne permet pas l'ouverture de plus de 15 fichiers en même temps, de plus parmi ceux-ci ne peuvent exister plus de 10 fichiers d'index.

4. Comme aucunes précautions n'ont été prises sur l'alignement des zones mémoire, le programme doit les gérer. L'alignement concerne le cas où une zone de caractères de longueur impaire est suivie d'une autre zone de caractères ou d'un entier.

5. On suppose que les courants des fichiers gérés par le module appelant sont correctement positionnés.

6. Il existe quatre fichiers différents. Pour chacun d'eux il existe une clé identifiante. De plus trois d'entre eux dispose d'une clé secondaire alors que le quatrième possède deux clés secondaires. Chacun des quatre fichiers est identifié par une famille 0, 1, 2, 3, qui correspond aux quatre informations du modèle m5.

7. On suppose qu'une mise-à-jour d'un record n'entraîne jamais de modification aux clés primaires et secondaires.

post-conditions :

1. ouverture

L'ouverture assure dans le cas du micro-ordinateur que tous les fichiers .dbf (contenant les informations) et les fichiers .ndx (servant d'index) de nom donné ont été ouverts. Ainsi quatre fichiers .dbf et neuf fichiers .ndx sont ouverts. Dans le cas du mini-ordinateur, les 4 fichiers sont ouverts.

2. fermeture

L'opération de fermeture assure dans le cas du micro-ordinateur que les 13 fichiers d'une base de nom donné sont fermés. Dans le cas du mini-ordinateur, les 4 fichiers sont fermés.

3. écriture

Dans le cas des fichiers de famille 0, 1, 2 le record, qui est passé en argument, est ajouté dans le fichier sans contrôle sur les clés. Les vérifications sur les clés sont effectuées par les modules de niveaux supérieurs. Dans le cas d'un fichier de famille 3, si un record de même clé secondaire existe et est actif c'est-à-dire qu'il n'a pas été supprimé logiquement, on refuse l'écriture du record. Dans tous les autres cas d'un fichier de famille 3 on écrit le record.

4. mise-à-jour

La mise-à-jour consiste à modifier les items n'appartenant pas aux clés des enregistrements

5. suppression

Cette opération entraîne la suppression logique d'un enregistrement. Tout accès par clé à cet enregistrement indiquera que le record est inactif signifiant par là cette suppression logique.

6. Lecture d'un record

Si une valeur de clé du record recherché doit être égale à celle passée en argument, la fonction cherche un record de même valeur de clé et qui n'a pas été supprimé logiquement. Si la valeur de clé du record recherché doit être supérieure ou égale à celle passée en argument, la fonction cherche le premier record non supprimé logiquement et dont la valeur de clé est bien la première valeur supérieure ou égale à celle spécifiée.

7. lecture du record suivant

On suppose que les accès par clè sont bien effectués et ainsi la fonction permet de sélectionner le record suivant par rapport à une valeur de clè spécifiée. Ce record doit ne pas avoir été supprimé logiquement ce qui peut nécessiter plusieurs accès.

3.2. Gestion des définitions de données

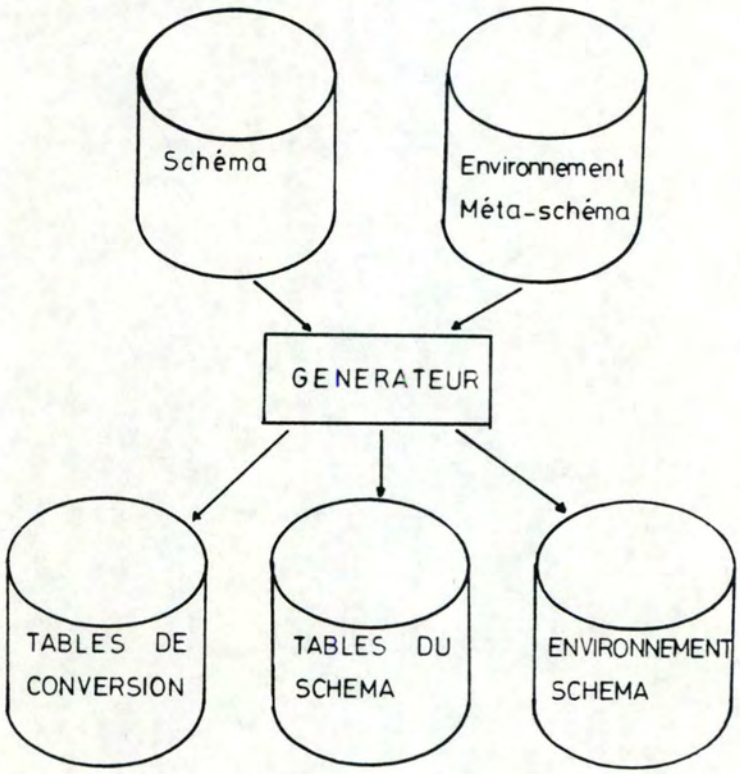
3.2.1. Générateur

Le principe du générateur est donc de fournir les moyens de traiter un schéma donné.

Pour que le générateur puisse fonctionner, il faut disposer d'une description du schéma sur lequel on désire travailler ainsi que de l'environnement du méta-schéma. Cet environnement consiste en un certain nombre de définitions et de structures correspondants au méta-schéma et qui permettent de comprendre et d'utiliser la description du schéma. Cette description du schéma aura été introduite dans les fichiers d'une base de données via le chargeur.

Disposant de ces deux élèments, le générateur permet de définir trois types d'informations:

1. Les tables du schéma qui sont groupées sous forme de quatre tables et qui renferment la description du schéma.
2. Les tables de conversion MAG/M5 qui sont la traduction du schéma externe en termes compris par le SGBD réel.
3. L'environnement du schéma qui est constitué d'informations permettant l'utilisation du schéma dans les termes décrits dans celui-ci.



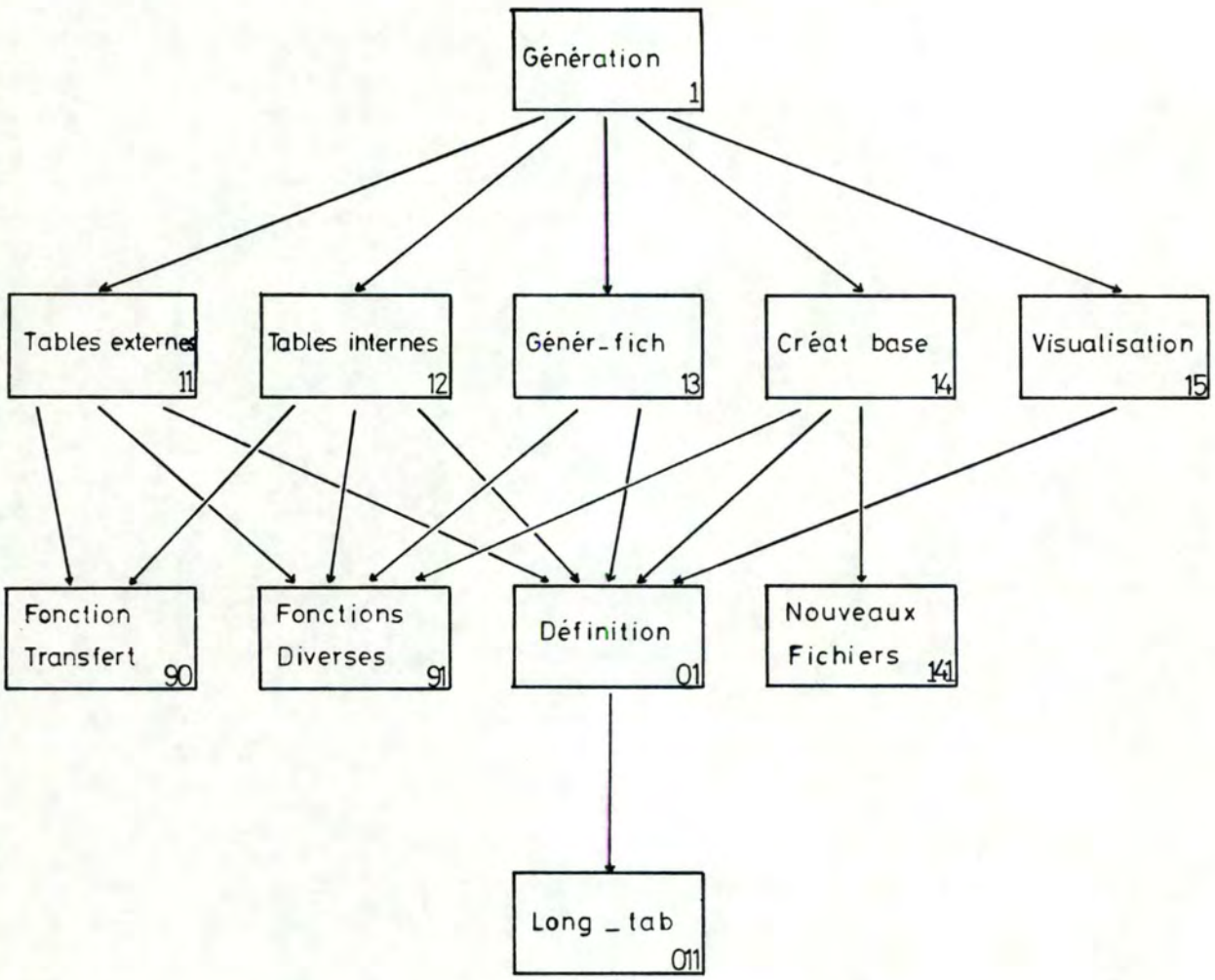
SCHEMA GENERAL

3.2.2. Architecture et principes algorithmiques

Préliminaire
- - - - - - -

La découpe reprenant toutes les fonctions, y compris leurs algorithmes, est fournie en annexe.

Découpe en modules
- - - - - - - - - - -



DECOUPE EN MODULES

Spécifications des modules

Module 1 : Génération

objectif :

Ce module a pour objectif de créer les tables externes et internes d'un schéma, de les mémoriser dans une base de données et de les visualiser à l'écran. De plus, ce module permet la création de fichiers contenant des informations nécessaires à la constitution de l'environnement du programmeur

nom du fichier :

génération.c

liste d'include :

/

liste des types :

/

pré-conditions :

Pour que la génération puisse s'effectuer, il faut qu'un schéma ait été chargé dans une base et que ce schéma respecte les régles de conformité du SGBD

post-conditions :

Les tables internes et externes sont créées, visualisées et chargées dans une base. De plus, les fichiers de l'environnement sont générés. Toutes ces informations correspondent au schéma fourni dans la base de départ.

fonction décrite :

main()

Module 11 : Tables externes

objectif :

Lorsqu'un schéma a été chargé dans une base de données, le module génère les tables du schéma.

nom du fichier :

tables_ext.c

liste des includes :

math.h
ida2.h
definition.ext
buf.ext

liste de types :

/

pré-conditions :

Pour que la génération puisse s'effectuer et respecter les spécifications, il faut que le schéma ait été correctement chargé dans une base.

conditions à respecter :

- les 4 premières lettres d'un nom d'entité sont identifiantes
- aucun rôle ou entité ne peut s'appeler TXT
- un accès dans un chemin n'est pas autorisé si le path-name = " "
- les connectivités acceptées sont 0-99999
  0-1
  1-1
- Dans le cas d'un chemin multi-origine on aura un rôle
  dont ROLE.MULTIPLE = "M"
  (les chemins multi-cibles ne sont pas acceptés)
- Les valeurs possibles de IKO.TYPE sont "I","K","O","IK","IO","KO","IKO"
- Les attributs d'une entité sont fournis dans n'importe quel ordre de position
- Une entité peut exister sans participer à un ROLE

- Le nombre d'entités définies est < 50
  de chemins définies est < 120
  d'IKO définies est < 15
  d'attributs définies est < 200
  Dans le cas où ces dimensions sont insuffisantes, il faut changer les valeurs dans le fichier donnant les longueurs des tables (long_tab.ext)

- Un attribut est défini "texte" par la valeur ATT.FORMAT = "T"

- Un identifiant est constitué de deux rôles ou d'un attribut dans un chemin

- Une entité est cible de maximum deux chemins obligatoires

- Il ne peut exister qu'un seul IKO dans un chemin

- Toutes les données introduites le sont en lettres majuscules

- COMPONENT.TYPE = atr ou rol

post-conditions :

Le programme se termine et crée les valeurs des tables externes du schéma.

Tables du schéma
- - - - - - - - - -
La description du schéma nécessite 4 tables.

ENTITY-T

E-NAME 30
E_CODE 3
E_LENGTH 3

E-NAME est le nom de l'entité

E-CODE est le code de l'entité

E-LENGTH est la longueur calculée en faisant la somme des longueurs des attributs

ATTRIBUTE

E-CODE	3
AT-NAME	30
AT-CODE	3
FORMAT	2
POSITION	2
POS-BYTE	2
LENGTH	3
NBR-DEC	2

Comme l'utilisateur peut définir un numéro d'ordre pour les attributs, la table reprenant ceux-ci sera triée par ordre de position croissante

E-CODE est le code de l'entité.

AT-NAME est le nom de l'attribut et celui-ci est identifiant dans l'entité

AT-CODE est le code identifiant de l'attribut dans l'entité

FORMAT vaudra C pour caractère
I pour integer
R pour réel
T pour indiquer le type texte

POSITION indique le numéro d'ordre de l'attribut parmi les autres attributs de l'entité

POS-BYTE indique la borne inférieure du début de la zone par rapport à l'ensemble du record (cette valeur sera calculée en prenant les longueurs et numéros d'ordre des attributs)

LENGTH donne la longueur totale de l'attribut y compris la virgule décimale si elle existe

NBR-DEC indique (pour les réels uniquement) le nombre de chiffres après la virgule décimale

```
ROLE
----
PATH_NAME 30
R-CODE     3
R-CODE-INV 3
MIN-CON    3
MAX-CON    5
E-CODE-M   3
I K O      1
IKO-NBR    3
```

PATH-NAME est le nom du chemin fourni dans le rôle. Dans le cas d'un chemin multi-origine, on a un seul path-name et un seul r-code mais on aura plusieurs records dans la table des ent-rôles

R-CODE est le code du rôle

R-CODE-INV est le code du rôle inverse

MIN-CON est la connectivité minimale
c'est-à-dire 0 ou 1

MAX-CON est la connectivité maximale
c'est-à-dire 1 ou 99999

E-CODE-M représente le code de l'entité du côté many de la relation.

IKO vaudra a s'il n'a pas d'identifiant
b si un attribut est identifiant dans un chemin
e si un rôle participe à un identifiant

IKO-NBR est le code de l'attribut participant à l'identifiant (uniquement dans le cas où IKO vaut b)

```
ENT-ROLE
--------
R-CODE     3
E-CODE-O   3
```

R-CODE est le code du rôle

E-CODE-O est le ou les codes des entités du côté origine de la relation. On ne définira ces origines que pour les rôles du côté one d'une relation one-to-many.

fonctions décrites :

```
tables_ext();
gen_attr_ext(ENT,long_ent);
gen_ent_ext(ENT,long_ent);
gen_role_ext(ROL);
rens_dir(ROL);
rech_code_inv(ROL);
identif(ROL);
ent_role(ROL);
code_lie_cible(ROL);
```

Module 12 : Tables internes

objectif :

L'objectif de ce module est de créer au départ des tables externes, l'ensemble des valeurs constituants les tables internes.

nom du fichier :

tables_int.c

liste d'include :

```
math.h
definition.ext
```

liste de types :

```
int EVE_BOOL;
char  EVE_POS_OWN[3];
char  EVE_NB_OWN[3];
```

pré-conditions :

Les tables externes sont créées et correctes par rapport au schéma fourni

post-conditions :

Les tables internes sont créées et rangées dans les 4 tables prévues à cet effet.

Tables internes

description des 4 tables:

INTERNAL-OBJECTS-CODE

cd	3
fam	3
type	3
length	3
pos_apt	3
nb_targ	3

cette table contient la description des objets sémantiques en termes d'objets du modèle M5

cd est le code sémantique de l'objet

fam indique le type d'objet M5

- object-0 = 0
- object-1 = 1
- object-2 = 2
- text = 3

type reprend le code sémantique de l'objet

length est la longueur totale de l'objet sémantique

pos_apt est la position du premier code dans la table des codes indiquant la première relation du groupe

nb_targ donne le nombre de relation dont l'objet est cible

INTERNAL-RELATIONS-TABLE

cd	3
cod_inv	3
cd_targ	3
impn	3
typ	3
iko	1
pos_iko	3
pos_own	3
nb_own	3

cd représente le code du chemin

cod_inv indique le code du chemin inverse

cd_targ est le code de la cible du chemin

impn indique le type de chemin du modèle M5
RO = 0 et le chemin inverse est 5
R1 = 1 et le chemin inverse est 6
R21 = 2 et le chemin inverse est 7
R22 = 3 et le chemin inverse est 8
OF = 4
chemin artificiel c'est-à-dire correspondant à un chemin non obligatoire = 9 et son inverse = 10

typ fournit le code du chemin si celui-ci est artificiel

iko permet de savoir s'il y a un identifiant dans la relation
a = pas d'iko
b = identifiant
c = clé
d = identifiant et clé
e = chemin intervenant dans l'identifiant qui n'est formé que de chemins

pos_iko donne la position de l'iko dans la table des iko

pos_own fournit la position du code du premier objet origine de la relation

nb_own indique le nombre d'origines à la relation

INTERNAL-IKO-TABLE
---
bi 3
length 3
impn 1

cette table renferme les informations concernant les identifiants d'objets sémantiques dans une relation

bi est la borne inférieure qui indique la position du premier caractère ou numérique de l'identifiant dans l'objet sémantique concerné

length est la longueur de la zone identifiante

impn vaut 0 (cette zone n'est pas utilisée)

INTERNAL-CD-TABLE
---
cd 3

cd est le code de l'objet ou de la relation
les blocs sont triés, on commence par les codes des chemins correspondants à pos_apt et nb_targ puis les autres codes correspondants à pos_own et nb_own de la table des relations. Lorsqu'on indique les codes des chemins, on commence toujours par ceux concernant les chemins obligatoires.

fonctions décrites :

```
tables-int();
det-famille(ent-crt,fam,pos-apt,nb-targ);
cherch-inv(roe-crt,inv-crt);
gen-oblig-codes-fam(ent-crt,fam);
gen-non-oblig(ent-crt,fam);
gen-obj-int(ent-crt,fam,pos-apt,nb-targ);
gen-rel-int(role-crt,inv-crt);
gen-iko(role-crt);
cherch-att(code-ent,code-att,position-att);
gen-origines(roe-crt,inv-crt);
txt-ob-rel();
txt-code();
```

Module 13 : Génération des fichiers

objectif :

Ce module a pour objectif de créer les valeurs correspondants à la création des fichiers constituants l'environnement du programmeur.

nom du fichier :

gener_fich.c

liste d'include :

```
stdio.h
math.h
definition.ext
```

liste de types :

```
FILE   *fopen(),*fp1;
FILE   *fopen(),*fp2;
FILE   *fopen(),*fp3;
FILE   *fopen(),*fp4;
```

pré-conditions :

Les tables externes et internes sont créées

post-conditions :

Les fichiers de l'environnement sont créées et les données qui sont contenues dans ces fichiers correspondent au schéma

environnement utilisateur

1) Pour chaque entité et chaque chemin d'accès, on crée une ligne d'un fichier du type suivant

#define <nom entité ou chemin> <code entité ou chemin>

2) Pour chaque entité, on crée dans un fichier les enregistrements du type suivant :

```
typedef struct i_<nom entité>_struct {
char <nom attribut >[<long attribut>];
char .....(même ligne pour chaque attribut)
}I_<nom entité >;

typedef struct ref_<nom entité>_str {
int REF,RTYPE,ILEN;
char <nom attribut>[<<long attribut>+1>];
char .....(même ligne pour chaque attribut)
}R_<nom entité>;
```

remarque : S'il n'y a pas d'attribut défini pour une entité, les deux structures sont générées avec un attribut technique c'est-à-dire

```
char ittech[1];
```

3) Un fichier contient deux premières lignes :

```
#include "ida2.h" et #include "text.h"
```

(ida2.h et text.h contiennent une suite de valeurs standards)

Par entité on crée une structure du type suivant permettant sa création par l'utilisateur:

a) si l'entité est cible obligatoire de 0 chemin et ne contient aucun attribut de type texte

```
C_<nom entité> (PARAM1)
R_<nom entité> *PARAM1;
{z_values.rectcode = <nom entité>;
PARAM1->ILEN=CST_MAXLEN;
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
}
```

b) si l'entité est cible d'1 chemin obligatoire et ne contient pas d'attribut de type texte

```
C_<nom entité 1> (PARAM1,PARAM2)
R_<nom entité 1> *PARAM1;
R_<nom entité 2> *PARAM2;
{z_values.rectcode = <nom entité 1 >;
PARAM1->ILEN=CST_MAXLEN;
pathlist[0]=<nom rôle 2>;
curlist[0]=(PARAM2->REF);
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
}
```

c) si l'entité est cible obligatoire de 2 chemins et ne possède pas d'attribut de type texte

```
C_<nom entité > (PARAM1,PARAM2,PATH1,PARAM4,PATH2)
R_<nom entité > *PARAM1;
R_RECORD *PARAM2,*PARAM4;
int PATH1,PATH2;
{z_values.rectcode = <nom entité 1 >;
PARAM1->ILEN=CST_MAXLEN;
pathlist[0]=PATH1;
curlist[0]=(PARAM2->REF);
pathlist[1]=PATH2;
curlist[1]=(PARAM4->REF);
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
}
```

d) si l'entité est cible obligatoire de 0 chemin et il existe au moins 1 attribut de type texte

```
C_<nom> (PARAM1,TEXTEP)
R_<nom> *PARAM1;
TEXTE *TEXTEP;
{char CAR;int I,J,K,M,CPTO,CPTC,FIN;R_TXT PTXT;
get_var(&(PTXT.REF));
{z_values.rectcode = <nom entité>;
PARAM1->ILEN=CST_MAXLEN;
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
if (dbstatus==ER_OK) {PTXT.GROUP[0]='a';
          PTXT.GROUP[1]='a'; PTXT.GROUP[2]='a';
CPTO=0;FIN=FALSE;CAR=*(TEXTEP->CH+CPTO);
            for(I=97;I<123 && FIN==FALSE;I++){
PTXT.SEQ_NB[0]=I;for(J=97;J<123 && FIN==FALSE;J++){
PTXT.SEQ_NB[1]=J;for(K=97;K<123 && FIN==FALSE;K++){
PTXT.SEQ_NB[2]=K;CPTC=0;while(FIN==FALSE && CPTC<80){
PTXT.TEXT[CPTC]=CAR;if(CPTO==TEXTEP->CURLEN) fin=TRUE;
else {CPTO++;CAR=*(TEXTEP->CH+CPTO);};CPTC++;
if (FIN==TRUE) for (M=CPTC;M<80;M++) PTXT.TEXT[M]=' ';
c_p_txt(&PTXT,PARAM1); if(dbstatus>ER_OK) FIN=TRUE;
};};};};free_var(&(PTXT.REF));}
```

e) si l'entité est cible obligatoire de 1 chemin et disposant d'au moins un attribut de type texte

```
C_<nom> (PARAM1,PARAM2,TEXTEP)
R_<nom> *PARAM1;
R_<nom> *PARAM2;
TEXTE *TEXTEP;
{char CAR;int I,J,K,M,CPTO,CPTC,FIN;R_TXT PTXT;
get_var(&(PTXT.REF));
{z_values.rectcode = <nom entité 1 >;
PARAM1->ILEN=CST_MAXLEN;
pathlist[0]=<nom rôle 2>;
curlist[0]=(PARAM2->REF);
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
if (dbstatus==ER_OK) {PTXT.GROUP[0]='a';
          PTXT.GROUP[1]='a'; PTXT.GROUP[2]='a';
CPTO=0;FIN=FALSE;CAR=*(TEXTEP->CH+CPTO);
            for(I=97;I<123 && FIN==FALSE;I++){
PTXT.SEQ_NB[0]=I;for(J=97;J<123 && FIN==FALSE;J++){
PTXT.SEQ_NB[1]=J;for(K=97;K<123 && FIN==FALSE;K++){
PTXT.SEQ_NB[2]=K;CPTC=0;while(FIN==FALSE && CPTC<80){
PTXT.TEXT[CPTC]=CAR;if(CPTO==TEXTEP->CURLEN) fin=TRUE;
else {CPTO++;CAR=*(TEXTEP->CH+CPTO);};CPTC++;
if (FIN==TRUE) for (M=CPTC;M<80;M++) PTXT.TEXT[M]=' ';
c_p_txt(&PTXT,PARAM1); if(dbstatus>ER_OK) FIN=TRUE;
};};};};free_var(&(PTXT.REF));}
```

f) si l'entité est cible obligatoire de deux chemins avec au moins un attribut de type texte

```
C_<nom> (PARAM1,PARAM2,PATH1,PARAM4,PATH2,TEXTEP)
R_<nom> *PARAM1;
R_RECORD *PARAM2,*PARAM4;
int PATH1,PATH2;
TEXTE *TEXTEP;
{char CAR;int I,J,K,M,CPTO,CPTC,FIN;R_TXT PTXT;
get_var(&(PTXT.REF));
{z_values.rectcode = <nom>;
PARAM1->ILEN=CST_MAXLEN;
pathlist[0]=PATH1;
curlist[0]=(PARAM2->REF);
pathlist[1]=PATH2;
curlist[1]=(PARAM4->REF);
transfert(&z_values.length,&(PARAM1->ILEN),
                    (PARAM1->ILEN)+sizeof(int));
sem(&dbstat,CREATE,0,&(PARAM1->REF),0,0,
                    pathlist,curlist,&z_values);
dbstatus=dbstat.errcode;
if (dbstatus==ER_OK) {PTXT.GROUP[0]='a';
          PTXT.GROUP[1]='a'; PTXT.GROUP[2]='a';
CPTO=0;FIN=FALSE;CAR=*(TEXTEP->CH+CPTO);
             for(I=97;I<123 && FIN==FALSE;I++){
PTXT.SEQ_NB[0]=I;for(J=97;J<123 && FIN==FALSE;J++){
PTXT.SEQ_NB[1]=J;for(K=97;K<123 && FIN==FALSE;K++){
PTXT.SEQ_NB[2]=K;CPTC=0;while(FIN==FALSE && CPTC<80){
PTXT.TEXT[CPTC]=CAR;if(CPTO==TEXTEP->CURLEN) fin=TRUE;
else {CPTO++;CAR=*(TEXTEP->CH+CPTO);};CPTC++;
if (FIN==TRUE) for (M=CPTC;M<80;M++) PTXT.TEXT[M]=' ';
c_p_txt(&PTXT,PARAM1); if(dbstatus>ER_OK) FIN=TRUE;
};};};};free_var(&(PTXT.REF));}
```

fonctions décrites :

```
gener_fich();
fich-def();
fich-struct();
fich-creat();
creer(nom,code);
cr-def1(nom);
cr-def2(nom);
cr-def3(nom);
cr-def4(nom);
un-path(e-code,nom-orig,role-orig);
cr-ref0(nom);
cr-ref1(nom,nom2,role2);
cr-ref2(nom);
cr-txt0(nom);
cr-txt1(nom,nom2,role2);
cr-txt2(nom);
majmin(a,b);
```

Module 14: Création base

objectif :

Les tables internes et externes ayant été créées, il faut les introduire dans les fichiers de la base de données.

nom du fichier :

creat-base.c

liste d'include :

```
m5struct.ext
tabconv.ext
definition.ext
refr.ext
Keepl.ext
```

liste de types :

```
define sys "000"
define object 0
define text 3
define of 4
struct m5struct z_val;
int z-recref;
int z-oref;
int z-pathlist[10]
int z-curlist[10]
```

pré-conditions :

les tables internes et externes sont chargées dans les tables partagées et les variables indiquent les bornes supérieures de ces tables

post-conditions :

Si la base de données existait, les anciennes tables sont remplacées par les nouvelles

Si la base de données n'existait pas, les tables internes sont crées dans de nouveaux fichiers dont le nom est fourni par l'utilisateur

fonction décrite :

```
creat_base();
```

Module 141 : Nouveaux fichiers

objectif :

Permet de créer l'ensemble des fichiers dbase 3 nécessaires à la constitution d'une base de données (cas du micro-ordinateur).

nom du fichier :

nouv_fich.c

liste d'include :

dbc.h

liste de types :

4 structures de fichiers DBC (Lattice C)

pré-conditions :

/

post-conditions :

Les 13 fichiers sont créées (4 DBF et 9 NDX), le nom donné constitue le préfixe de chaque nom de fichier.

fonction décrite :

nouv_fich(nom)

Module 15 : Visualisation.

objectif :

Lorsque les tables du schéma et les tables internes sont créées, on peut par ce module visualiser leurs contenus.

nom du fichier :

visualisation.c

liste d'includes :

definition.ext
m5struct.ext
tabconv.ext

liste de types :

/

pré-conditions :

- Toutes les tables sont chargées dans les tables définies dans le fichier partagé
- Pour chaque table, une variable indique la borne supérieure

Post-conditions :

L'ensemble des contenus des tables a été visualisé à l'écran

fonction décrite

visualisation();

Module 90 : Fonction de transfert

objectif :

Transférer une chaîne de caractères dans une autre en mettant des 0 davant la chaîne si la longueur da la chaîne émettrice n'est pas suffisante pour combler la chaîne réceptrice.

nom du fichier :

transf2.c

liste d'include :

/

liste de types :

/

pré-conditions :

La longueur de la chaîne émettrice est plus petite ou égale à la longueur de la chaîne réceptrice. La chaîne initiale est une chaîne de caractères.

post-conditions :

La chaîne de caractères de la zone émettrice est transférée dans la chaîne de la zone réceptrice et celle-ci est terminée par le caractère 0 ce qui lui donne bien le statut de chaîne de caractères.

fonction décrite :

transf2(b1,b2,length)

Module 91 : Fonctions diverses

objectif :

Permet de fournir au programme principal, une série de fonctions

nom du fichier :

itoa.c

liste d'include :

/

liste de types :

/

pré-conditions :

les arguments (s'ils existent) sont corrects

post-conditions :

s contient la chaîne de caractères correspondant à l'entier n (fonction itoa)

s renvoie la chaîne de caractères retournée (fonction reverse)

s renvoie la longueur de la chaîne de caractère (fonction strlen)

fonctions décrites :

itoa(n, s)

reverse(s)

strlen(s)

Module 01 : Définition

objectif :

Ce module contient les structures des 8 tables à créer ainsi que des variables permettant le contrôle de leurs dimensions. De plus, on y définit les variables de référence

nom du fichier :

définition.c

liste d'include :

long_tab.ext

liste de types :

/

pré-conditions :

L'espace des variables ne peut excéder 64K. Les tables ici définies ne pourront avoir une dimension trop grande.

post-conditions :

- les variables sont créées
- les structures sont créées
- les variables servant aux références sont définies

fonction décrite :

/

Module Oii : Longueur des tables

objectif :

Ce module permet de définir la longueur des tables externes et internes qui seront utilisées en mémoire centrale. La dimension de ces tables dépend du schéma de la base de données.

nom du fichier :

long_tab.ext

liste d'include :

/

liste de types :

/

pré-conditions :

Les tailles des tables ne peuvent pas être trop importante pour éviter de dépasser la taille de la mémoire.

post-conditions :

Les tailles des tables sont suffisantes pour le schéma décrit et ne dépassent pas la taille de la mémoire.

fonction décrite :

/

3.2.3. Tests

Puisque les tables de conversion ainsi que l'environnement sont nécessaires à l'exploitation d'une base de données, le test consiste à remplacer ces différents éléments chargés manuellement par les mêmes composants générés.

Initialement le chargement dans une base de données d'un schéma nommé BIBLIO reprenant la description de la gestion d'une bibliothèque a été testé mais en disposant d'un environnement et de tables chargés manuellement. Pour charger ce schéma on a besoin des tables internes du méta-schéma ainsi que de l'environnement du méta-schéma.

Différentes étapes ont été réalisées pour tester le générateur.

1. Création d'un fichier, dans le langage défini pour le chargement d'un schéma, reprenant la description du méta-schéma.

2. Création d'un fichier reprenant la description du schéma de la bibliothèque.

3. Disposant de l'environnement du méta-schéma et du fichier reprenant la description du méta-schéma (ces deux éléments étant créées manuellement) charger une base de données contenant le méta-schéma (et dont les tables sont créées manuellement) grâce au fichier créé au point 1.

4. Ayant une base de données décrite ci-dessus, générer les tables du méta-schéma dans une nouvelle base de données et créer les fichiers de l'environnement du méta-schéma. Appelons cette base, la base BIBLIO.

5. Maintenant que l'on dispose d'une base de données contenant des tables générées du méta-schéma et de l'environnement généré du méta-schéma, on peut charger le fichier dans le langage défini, créé au point 2, dans la base BIBLIO. Le bon déroulement du chargement prouve que les tables et l'environnement générés correspondent aux spécifications.

Conclusions

L'objectif de ce mémoire était triple. Premièrement nous nous sommes attachés au transport de l'atelier logiciel orienté bases de données. Ce transport assure maintenant une portabilité très grande du logiciel et permet de décentraliser l'aide à la conception d'une base de données si on se place dans le cadre du projet IDA. La deuxième étape du travail nous a permis de construire le générateur qui permet au départ d'un schéma de base de données d'en déduire des tables ainsi qu'un environnement permettant l'exploitation de cette base de données par l'atelier. Enfin la troisième partie du travail a été la constitution de ce présent document qui permet de retracer les différentes étapes du travail mais surtout de replacer ce projet dans le cadre plus général des bases de données.

Lors des différentes étapes de travail, nous avons rencontré certaines difficultés. Tout d'abord le fait que le projet d'atelier logiciel évolue sans cesse ne permet pas d'assurer que les informations reçues soient définitives. A tout moment de l'évolution du projet, des modifications sont apportées ce qui nécessite de nombreux changements dans les spécifications des outils. La deuxième difficulté est consécutive à la première et se rapporte aux spécifications. En effet, le fait que le produit évolue constamment implique un retard dans la documentation parce que sa mise-à-jour n'est pas toujours considérée comme urgente. Cette difficulté de disposer d'une information précise à tout moment entraîne des erreurs dans le développement des outils.

L'avenir de cet atelier logiciel consiste maintenant à réaliser différents outils entourant la base de données et facilitant l'aide à la conception. Pour l'utilisateur, ce sont les outils qui lui sont proposés qui constituent l'attrait du logiciel.

Bibliographie

(AD-DEL) M. ADIBA et C. DELOBEL: Base de données et systèmes relationnels

(AVL-84) A. VAN LAMSWEERDE: Cous de Méthodologie de développement de logiciels. Seconde licence et maîtrise en informatique.

(BOD-PIG-83) F. BODART, Y. PIGNEUR: Conception assistée des applications informatiques
1.Etude d'opportunité et analyse conceptuelle

(BOV-85) BOVESSE: Les dictionnaires de données

(CAD-86) M. CADELLI: Programmation sur la base de données de l'atelier.

(CHAR-MUL-85) C. CHARLOT et I. MULLER: Contribution à l'atelier logiciel de conception de bases de données: études de transformations de schémas.

(CLAR-81) A. CLARINVAL: Comprendre, connaître et maîtriser le Cobol

(CUR-DIEC) E. MARTIN DIECKMANN et ROBERT M. CURTIE: A survey of data dictionaries

(DD-84) Data Analysis and Data Dictionary/Directory Systems April 4-6.84 London Frost & Sullivan Ltd

(DDSWP-82) Data Dictionary Systems Working Party (DDSWP) Journal of Development Summer 1982

(DEL-86) A. DELCOURT: Spécification du fichier de chargement d'une base

(DEL-VH-85) A. DELCOURT et B. VANHOUTTE: Dossier technique du SGBD IDA2

(DEL-VH-85) A. DELCOURT et B. VANHOUTTE: Dossier d'utilisation du SGBD IDA2

(HAI-86) J-L. HAINAUT: Conception assistée des applications informatiques
2. Conception de la base de données

(HAI-86a) J-L. HAINAUT: Atelier de conception d'applications sur bases de données : Schémas de la base de spécifications.

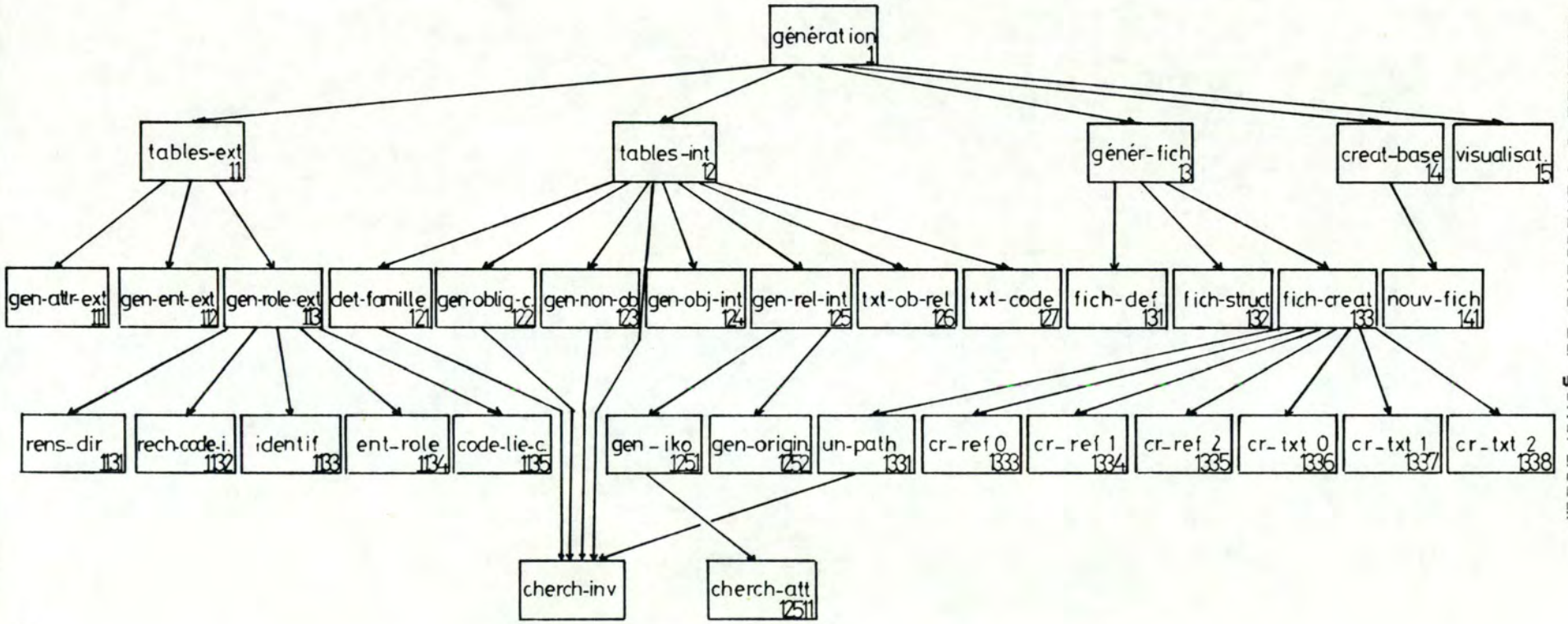
(KER_RIT_78) BRIAN W. KERNIGHAN ET DENNIS M. RITCHIE: The C programming language.

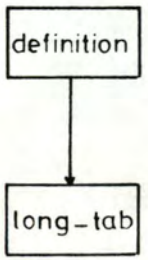
(LATT-82) LATTICE C Manual LATTICE 8086/8088 C Compiler Lifeboat associates (Released May 25,1982)

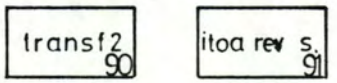
(LATT-85) Lattice DBC Library (dbase 3 option) Lattice Inc (Released January 7,1985)

(MA-82) ALAN MAYNE : Central role of dictionary systems in informations resource management

(MAR-75) JAMES MARTIN : Computer Data Base Organization


génération
1
tables-ext
11
tables-int
12
génér-fich
13
creat-base
14
visualisat.
15
gen-attr-ext
111
gen-ent-ext
112
gen-role-ext
113
det-famille
121
gen-oblig-c.
122
gen-non-obl
123
gen-obj-int
124
gen-rel-int
125
txt-ob-rel
126
txt-code
127
fich-def
131
fich-struct
132
fich-creat
133
nouv-fich
141
rens-dir
1131
rech-code-i
1132
identif
1133
ent-role
1134
code-lie-c.
1135
gen-iko
1251
gen-origin
1252
un-path
1331
cr-ref 0
1333
cr-ref 1
1334
cr-ref 2
1335
cr-txt 0
1336
cr-txt 1
1337
cr-txt 2
1338
cherch-inv
cherch-att
12511


definition
long-tab


transf2
90
itoa rev s.
91

Spécifications des fonctions

prérequis:

Dans la description des différentes fonctions, nous avons spécifié entre autres les différents algorithmes. C'est la langage naturel qui dans la plupart des cas a été utilisé mais nous avons fait aussi appel aux notions du langage ADL dont nous reprenons ci-après les éléments nécessaires. La langage ADL permet l'expression d'algorithmes dans la cadre de la démarche de conception des traitements (HAI-86).

Les notions utilisées par la suite sont:

a:=A(B:C)	a est une valeur de type A liée à C par le chemin B
for...do	équivaut à pour...faire
endfor	indique la fin de la boucle for
exit	provoque la sortie prématurée d'une boucle
if no-C	équivaut à "s'il n'y a pas d'élément de type C"
endif	indique la fin de la boucle if

Fonction 1 : main

objectif :

Cette fonction permet de créer les tables externes du schéma dans la base, de plus elle crée les tables internes qui seront rangées aussi dans les fichiers de la base et enfin un ensemble de trois fichiers constituants un environnement pour le programmeur. Toutes les tables seront visualisées à l'écran après avoir été chargées dans les fichiers d'une base.

nom du fichier :

generation.c

argument de la fonction :

/

variables globales :

/

variables locales :

/

algorithme :

-génération des tables externes
-génération des tables internes
-génération des fichiers constituants l'environnement du programmeur
-création d'une base reprenant les tables internes et externes
-visualisation des tables externes puis internes

Fonction 11 : Tables externes

objectif :

Cette fonction permet de créer les tables externes correspondant au schéma. Cette création s'effectue par parcours du schéma.

nom du fichier :

tables_ext()

argument de la fonction :

/

variables globales :

/

variables locales :

```
variable de référence system SYS
                      schéma SCH
                      link LK
                      rôle ROL
                      entity-t ENT
int err;              /* traitement des erreurs */
char *long_ent[3];    /* longueur d'une entité  */
int trouve-sch;       /* schéma trouve ou non   */
char name[6];         /* nom de la bd           */
char name-sch[31];    /* nom du schéma          */
```

algorithme :

- ouverture de la base et initialisation de la recherche
  dans le schéma
    - saisie du nom de la bd contenant le schéma
    - ouverture de cette bd
    - definition des variables de référence
    - saisie du nom du schéma concerné par la génération
    - acces au système puis au schéma
- générer l'ensemble des valeurs pour la table des entités et pour celle des attributs
  for ENT:= ENTITY-T(S-E:SCH) do
    - génération des valeurs dans la table des attributs par appel à gen-attr-ext
    - génération des valeurs dans la table des entités par appel à gen-ent-ext
  endfor
- générer les valeurs adans les tables des rôles et des ent-rôles
  for LK:=LINK(S-L:SCH) do
    for ROL:=ROLE(L-R:LK) do
      générer les valeurs dans les tables des rôles et des ent-rôles
    endfor;
  endfor;
- fermeture de la base et suppression du statut de référence aux variables de référence

Fonction 111 : Gen-attr-ext

objectif :

Cette fonction permet de créer l'ensemble des valeurs pour la table des attributs relativement à une entité définie. Les différentes valeurs seront classées par ordre croissant de position de l'attribut.

nom du fichier :

tables_ext.c

variables globales :

/

variables locales :

```
variable de référence attribute ATT;
int lg;            /* longueur courante de l'entité    */
int lng;           /* longueur d'un attribut           */
int t-att-ent;     /* jauge de la table intermédiaire */
int pos-at-prec;   /* position attribut précédent      */
int pos-att;       /* position attribut courant        */
int i;             /* indice table des attributs       */
int j;             /* indice table des attributs       */
int sauv-ind;      /* sauvetage indice des attributs   */
int init-att;      /* indice premier attribut          */
int pos-byte;      /* position en byte de l'attribut   */
char p-bytes[3];   /* pos-byte sous forme de caract.   */
char s-ent[4];     /* long ent sous forme caractères   */
struct attribute p-att-ent[25];
                   /* table intermédiaire d'attributs */
```

argument de la fonction :

ENT (entrée) indique la référence de l'entité traitée
LENGTH (sortie) indique la longueur totale de l'entité par sommation des longueurs des attributs

algorithme :

```
- définir une variable de référence attribute
- t_att_ent = 0; (jauge de la table intermédiaire)
- for A:=ATTRIBUTE(EL-AT:ENT)
    - mettre les valeurs de l'attribut dans la table
      intermédiaire (e-code, at-code, at-name, format
      position, length, nbr-dec)
   - incrémenter jauge de la table intermédiaire
     t_att_ent = t_att_ent + 1;
- (tri des attributs par ordre croissant de position)
  pos_at_préc = 0; (position attribut précédent)
  sauv_ind = 99; (indice sauvé)
  init-att = top-attr;
  for (i=0;i<t_att_ent;i++)
    - pos_att = 99; (position attribut courant)
    - for (j=0;j<t_att_ent;j++)
        - if (p_att_ent[j] < pos_att &&
                           >= pos_at_prec &&
              sauv_ind != j)
                  - sauver indice j
                  - pos_att = position trouvee
    - recopier les valeurs de la table
      intermédiaire vers la bonne table
    - incrementer jauge des attributs
    - sauver position dans pos_at_prec
- calcul de length et de pos_byte
  lg = 0; pos_byte = 0;
  for (i=init-att;i<top-attr;i++)
      si c'est le premier attribut pos_byte = 0
      sinon pos_byte = length;
      lg = lg + longueur de l'attribut
  transférer la longueur de l'entité
- libérer la variable de référence
```

Fonction 112 : Gen_ent_ext

objectif :

Cette fonction permet d'inscrire dans la table des entités toutes les valeurs relatives à l'entité traitée.

nom du fichier :

tables_ext.c

variables globales :

/

variables locales :

/

arguments de la fonction :

ENT (entrée) référence à l'entité que l'on traite
LENGTH (entrée) longueur de l'entité

algorithme :

- copier le ENT.E_NAME dans la zone de la table
- copier le ENT.E_CODE dans la zone de la table
- copier la longueur LENGTH dans la zone de la table
- incrémenter la jauge de la table des entités

Fonction 113 : Gen_role_ext

objectif :

Cette fonction permet d'inscrire pour un rôle déterminé toutes les informations dans la table des rôles et dans celle des ent-rôles pour autant que le rôle n'ait pas encore été traité précédemment

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

```
int rech-role;      /* indice table des roles          */
int trouve-role;    /* booléen indiquant si le role */
                    /* existe déjà                     */
```

argument de la fonction :

ROL (entrée) référence au rôle que l'on doit traiter

algorithme :

- rech-role = 0;
- trouve-role = 0;
- voir si le code du rôle existe déjà dans la table des rôles si oui trouve-role = 1;
- si le rôle n'a pas encore été traité (trouve-role=0):
  - générer les renseignements directement accessibles
  - générer le code du rôle inverse
  - déterminer si le rôle participe à un identifiant
    et si nécessaire le code de l'attribut participant
  - définir les codes des origines du chemin
  - définir le code lié à la cible d'un chemin
  - incrémenter la jauge de la table des rôles

Fonction 1131 : Rens_dir

objectif :

Générer les renseignements, directement connus du rôle, dans la table des rôles

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

/

argument de la fonction :

ROL (entrée) référence au rôle que l'on doit traiter

algorithme :

- transférer dans la ligne de la table des rôles
  - path-name = ROL.PATH_NAME
  - r-code = ROL.R_CODE
  - min-con = ROL.MIN_CON
  - max-con = ROL.MAX_CON

Fonction 1132 : Rech_code_inv

objectif :

Rechercher le code inverse du rôle que l'on est en train de traiter

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

variable de référence rôle ROL2
link LK

argument de la fonction :

ROL (entrée) référence au rôle que l'on traite

algorithme :

- définir les deux variables de référence
- L:=LINK(R-L:ROL)
- for R:=ROLE(L-R:L)
  if (ROL.R_CODE != R.R_CODE) exit R;
  endfor
- transferer R.R_CODE dans la zone r-code-inv de la table des rôles
- libérer les deux varaibles de référence

Fonction 1133 : identif

objectif :

Déterminer si le rôle participe à un identifiant et éventuellement le code de l'attribut participant

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

variable de référence component COMP
id-key-ord IKO
attribute ATT

argument de la fonction :

ROL (entrée) référence au rôle que l'on traite

algorithme :

```
- définir les variables de référence
- C:=COMPONENT(RA-C:ROL)
- if no-C then
    { mettre la valeur "a" dans la zone iko de la table
      des rôles
      mettre la valeur "000" dans la zone pos-iko de la
      table des rôles
    }
  else
    { IK:=IKO(C-IKO:C);
      booleen-atr = false;
      for C:=COMPONENT(IKO-C:IK)
          if (C.TYPE = atr)
             then booleen-atr = true; exit C;
          endif
      endfor
      if (booleen-atr = true)
       { A:=ATTRIBUTE(C-RA:COMP);
         mettre A.AT-CODE dans la zone iko-code de la
              table des rôles
         mettre "b" dans la zone iko de la table des
         rôles
       }
      else
       { mettre "e" dans la zone iko de la table des
         rôles
         mettre "000" dans la zone iko-code de la table
             des rôles
       }
    }
- libérer les variables de référence
```

Fonction 1134 : Ent_role

objectif :

Définir le ou les codes origines des chemins. Dans le cas des chemins one-to-many (côté one de la relation) on inscrira le code de l'entité ou des entités origines dans la table

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

```
variables de référence role ROL1
                        link LK
                        entity-t ENT
```

argument de la fonction :

ROL (entrée) référence au rôle à traiter

algorithme :

```
- définir les variables de référence
- si ROL.MAX_CON <= 1 exit;
  si ROL.MULTIPLE = E
   { inscrire ROL.R_CODE dans la table ent-rol (r-
     code)
     inscrire "EVE" dans la table ent-rol (e-code-o)
     incrémenter jauge de la table ent-rol
   }
  sinon
     FOR L:=LINK(R-L:ROL)
       FOR ROL1:=ROLE(L-R:L)
         IF (ROL1.R_CODE == ROL.R_CODE)
            FOR E:=ENTITY(R-E:ROL1)
                inscrire ROL.R_CODE dans la table
                        ent-rol
                inscrire E.E_CODE dans la table ent-rol
                incrémenter jauge de la table ent-rol
            ENDFOR
         ENDIF
       ENDFOR
     ENDFOR
  - libérer les variables de référence
```

Fonction 1135 : Code-lié-cible

objectif :

Cette fonction génère le code côté many d'une relation.
On n'inscrira une valeur uniquement dans le cas du rôle côté one de la relation.

nom du fichier :

tables-ext.c

variables globales :

/

variables locales :

```
variables de référence role ROL1
                       link LK
                       entity-t ENT
```

argument de la fonction :

ROL (entrée) référence au rôle traité

algorithme :

```
- définir les variables de référence
- si max-con du rôle courant (role-crt) dans la table
  des rôles a une connectivité maximale <= 1
     alors e-code-m dans la table des rôles = "000"
  sinon
   FOR L:=LINK(R_L:ROL)
     FOR ROL1:=ROLE(L_R:L)
       IF (ROL1.R_CODE != ROL.R_CODE)
         FOR E:=ENTITY(R_E:ROL1)
             inscrire E.E_CODE dans e-code-m pour
                le rôle dont l'indice est role-crt
         ENDFOR
       ENDIF
     ENDFOR
   ENDFOR
- libérer les variables de référence
```

Fonction 12 : Tables_int

objectif :

L'objectif de cette fonction est de créer les tables internes du schéma au départ des tables externes

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
int fam;        /* famille de l'objet                 */
int pos-apt;    /* position premier code de chemin    */
int nb-targ;    /* nombre de cibles                   */
int inv-crt;    /* indice du rôle inverse             */
int role-crt;   /* indice du rôle traité              */
int ent-crt;    /* indice dans la table des entités   */
```

argument de la fonction :

/

algorithme :

(création d'une entrée dans la table des relations pour chaque entrée dans la table des rôles)
- pour chaque ligne de la table des rôles
    - générer le code du rôle dans la table des relations comme étant égal au code du rôle dans la table des rôles
    - incrémenter la jauge de la table des relations

(création de toutes les lignes dans la table des objets et dans celle des codes, représentant les chemins pour lesquels les entités sont cibles)
- pour chaque ligne de la table des entités
    - déterminer la famille
    - générer les codes des chemins obligatoires
    - générer les codes des chemins non obligatoires
    - générer une ligne dans la table des objets

(création des valeurs dans la table des relations et dans celle des ikos)
- pour tous les éléments dans la table des rôles dont la connectivité maximale est > 1
    - chercher l'indice du code du rôle inverse
    - générer les valeurs dans les tables des relations et des ikos

(informations relatives aux textes)
- création d'une ligne spéciale dans la table des relations, des objets et des ikos avec les valeurs correspondantes dans la table des codes
- créer les valeurs dans la table des codes, pour définir les entités origines du chemin OF (modèle M5)

Fonction 121 : Det-famille

objectif :

Cette fonction détermine le nombre de chemins dont l'entité est cible et parmi ceux-là ceux dont elle est cible obligatoire

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
int nbr-cible; /* nombre de cible                    */
int nbr-oblig; /* nombre de chemins obligatoires */
int inv-crt;   /* indice du role inverse           */
int role-crt;  /* indice du role courant           */
```

arguments de la fonction :

ent_crt (entrée) indice de la ligne traitée dans la table des entités
fam (sortie) famille de l'entité (voir schéma M5)
pos_apt (sortie) position dans loa table des codes du premier code de chemin dont l'entité est cible
nb_targ (sortie) nombre de chemins dont l'entité est cible

algorithme :

- mettre jauge des codes dans pos-apt
- mettre nb-targ et fam à 0
- mettre le nombre de cible et le nombre de chemins obligatoires à 0 (nbr-cible et nbr-oblig)
- pour chaque ligne de la table des rôles dans laquelle
  le e-code-m = le code de l'entité dont l'indice est donné par ent-crt
  - accès à la ligne dans rôle dont le code = code inverse de la ligne role-crt que l'on traite
  - si min-con > 0
    nbr-oblig = nbr-oblig + 1;
  - si max-con = 1
    nbr-cible = nbr-cible + 1
- mettre nbr-cible dans nb-targ
- si nbr-oblig = 0 alors famille (fam) = 0
  si nbr-oblig = 1 alors famille (fam) = 1
  si nbr-oblig = 2 alors famille (fam) = 2

Fonction 122 : Gen_oblig_codes_fam

objectif :

Cette fonction génère les codes dans la table des codes, indiquants les chemins dont l'entité est cible. De plus, il génère les codes dans la table des relations indiquants les familles des rôles relativement aux chemins dont l'entité est cible et ceci uniquement pour les rôles obligatoires.

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
int inv-crt;          /* indice du role inverse          */
int bool-prem-role;   /* indique si c'est le premier     */
                      /* role    que l'on traite         */
int role-crt;         /* indice du role traité           */
```

arguments de la fonction :

ent_crt (entrée) indice de la ligne traitée dans la table des entités
fam (entrée) famille de l'entité

algorithme :

- bool-prem-role = 1;
- pour chaque ligne de la table des rôles dans laquelle le e-code-m = code de l'entité dont l'indice est donné par ent-crt (ce sera toujours un rôle côté one d'une relation one-to-many puisque le e-code-m n'est utilisé que pour les rôles ones) (soit le rôle r1)
  - accès à la ligne correspondant au code du rôle inverse (soit rôle r2)
  - si r2.min-con > 0
    - si fam = 2
      - si c'est le premier rôle qu'on traite (bool-prem-role = 1)
        -mettre 007 dans la famille de r2
        -mettre 002 dans la famille de r1
        -créer ligne dans la table des codes avec la valeur r1
        -mettre "000" dans typ pour r2
        -mettre "000" dans typ pour r1
        -incrémenter jauge des codes
        -boo-prem-role = 0;
      - sinon
        -mettre 008 dans la famille de r2
        -mettre 003 dans la famille de r1
        -créer ligne dans la table des codes avec la valeur r1
        -mettre "000" dans typ pour r2
        -mettre "000" dans typ pour r1
        -incrémenter jauge des codes
    - si fam = 1
      -mettre 006 dans la famille de r2
      -mettre 001 dans la famille de r1
      -créer ligne dans la table des codes avec la valeur r1
      -mettre "000" dans typ pour r2
      -mettre "000" dans typ pour r1
      -incrémenter jauge des codes

Fonction 123 : Gen-non-oblig

objectif :

Cette fonction permet de générer dans la table des codes ainsi que dans la table des relations les valeurs indiquants les familles des rôles non obligatoires ainsi que les codes des entités correspondantes.

nom du fichier :

tables_int.c

variables globales :

/

variables locales :

```
int bool-prem-role; /* indique si c'est la premier
                       role que l'on traite          */
int role-crt;      /* indice du role triaté         */
int inv-crt;       /* indice du role inverse        */
```

arguments de la fonction :

ent_crt (entrée) indice de la ligne traitée dans loa table des entités
fam (entrée) famille de l'entité

algorithme :

- bool-prem-role = 1;
- pour chaque ligne de la table des rôles dans laquelle e-code-m = code de l'entité dont l'indice est donné par ent-crt
  - accès au rôle inverse (soit r2)
  - si min-con = 0 et max-con = 1
    - si fam=0 et premier passage dans la boucle
      - mettre 005 dans la famille de r2
      - mettre 000 dans la famille de r1
      - créer une ligne dans la table des codes
      - incrémenter jauge des codes
      - mettre "000" dans typ pour r2
      - mettre "000" dans typ pour r1
      - bool-prem-role = 0
  - si min-con = 0
    - mettre 010 dans la famille de r2
    - mettre 009 dans la famille de r1
    - créer une ligne dans la table des codes
    - répéter code du rôle one dans type de r2
    - répéter code du rôle one dans type de r1
    - incrémenter jauge des codes

Fonction 124 : Gen-obj-int

objectif :

Le but de cette fonction est d'inscrire dans la table objets la ligne correspondant à l'entité dont l'indice est fourni dans ent_crt

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
char s[3]; /* chaine intermédiaire pour la fonction
              itoa */
```

arguments de la fonction :

ent_crt (entrée) indice de la ligne traitée dans la table des entités
fam (entrée) famille de l'entité
pos_apt (entrée) position du premier code de chemin dont elle est cible
nb_targ (entrée) nombre de chemin dont l'entité est cible

algorithme :

- inscrire le code de l'entité dans la table des objets
- inscrire la famille dans la table des objets
- inscrire pos-apt dans la table des objets
- inscrire nb-targ dans la table des objets
- inscrire le type de l'entité qui est égal au code
- inscrire length de l'entité
- incrémenter top-obj la jauge de la table des objets

Fonction 125 : Gen-rel-int

objectif :

Le but de cette fonction est de créer l'ensemble des valeurs dans la table des relations et dans celle des ikos pour ce qui concerne un rôle et son inverse.

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

/

arguments de la fonction :

role_crt (entrée) position de la ligne traitée dans la table des rôles

inv_crt (entrée) position de la ligne correspondant au rôle inverse de celui défini dans la table des rôles à l'indice rôle_crt

algorithme :

```
(traitement du role-crt)
-  copier le r-cod-inv de la table des rôles vers
   cod-inv de la table des relations
- copier e-code-m (rôles) vers cd-targ (relations)
- copier iko (rôles) vers iko (relations)
- copier "000" vers pos-iko (relations)

(identifiant)
- si code de l'iko est "b"
      appeler la fonction gen-iko

(origines)
- générer les codes origines des chemins

(traitement du rôle inverse)
- copier r-cod-inv (rôles) vers cod-inv (relations)
- copier e-code-m (rôles) vers cd-targ (relations)
- copier "a" vers iko (relations)
- copier "000" vers pos-iko (relations)
```

Fonction 1251 : Gen-iko

objectif :

Le but de cette fonction est de créer l'ensemble des valeurs dans la table des ikos pour l'attribut qui participe à un identifiant

nom du fichier :

tables_int.c

variables globales :

/

variables locales :

```
int position-att; /* position de l'attribut       */
char s_iko[3];    /* chaine pour la fonction itoa */
```

argument de la fonction :

role_crt (entrée) indice dans la table des rôles, du rôle que l'on traite

algorithme :

- mettre la valeur de la jauge des ikos dans pos-iko de la table des relations
- rechercher la position d'un attribut de code donné
- transférer length de la table des attributs vers la table des ikos
- transférer pos-byte de la table des attributs vers la zone bi de la table des ikos
- mettre "0" dans la zone impn de la table des ikos
- incrémenter la jauge de la table des ikos

Fonction 12511 : Cherch-att

objectif :

Connaissant le code d'un attribut et de l'entité, fournir la position de celui-ci dans la table des attributs.

nom du fichier :

tables_int.c

variables globales :

/

variables locales :

/

arguments de la fonction :

code_ent (entrée) code de l'entité
code_att (entrée) code de l'attribut
position_att (sortie) position de l'attribut dans la table des attributs

algorithme :

- position_att=0;
- tant que le code de l'attribut dans la table des attributs est différent du code-att et que le e-code de la table des attributs est différent du code-ent
  - incrémenter position-att

Fonction 1252 : Gen-origines

objectif :

Création des codes dans la table des codes, définissant les entités origines des chemins

nom du fichier :

tables-int.c

variables globales :

```
int EVE_BOOL;          /* booleen pour voir si on a
                          déjà rencontrer le cas EVE */
char EVE_NB_OWN[3]; /* nombre d'origines du chemin
                          EVE */
char EVE_POS_OWN[3];/* position de la première origine
                          du chemin EVE */
```

variables locales :

```
char s-cd[3];          /* chaine pour la fonction itoa */
char s-orig[3];        /* chaine pour la fonction itoa */
int i;                 /* recherche table ent-rol      */
int j;                 /* recherche table des entités  */
int nbr-orig;          /* nombre d'origines            */
```

arguments de la fonction :

role_crt (entrée) indice de la ligne traitée dans la table des rôles
inv_crt (entrée) indice de la ligne du rôle inverse

algorithme :

```
- si EVE.BOOL = 0 mettre top-cd dans EVE_POS_OWN
- mettre top_cd dans la zone pos-own à l'indice role-
  crt
- mettre top_cd dans la zone pos-own à l'indice inv-
  crt
- nbr-orig = 0
- for (i=0;i<top_ent_rol;i++)
    - if (r-code dans table ent-rol = r-code dans la
         table des rôles indice role-crt)
            if (e-code-o dans table ent-rol = EVE)
                 if (EVE_BOOL = 0)
                    -  pour toutes les entités
                       existantes
                    - inscrire le code de l'entité
                    - incrémenter jauge des codes
                    - nbr-orig = nbr-orig + 1;
                    - EVE_BOOL = 1;
                    - EVE_NB_OWN = nbr_orig
                 else
                    - mettre EVE_POS_OWN dans pos-own
                      pour les indices role-crt et
                      inv-crt
                    - nbr_orig = EVE_NB_OWN
            else
               inscrire e-code dans la table
               incrémenter jauge des codes
               nbr-orig = nbr-orig + 1
- inscrire nbr-orig dans nb-own indice role-crt
- inscrire nbr-orig dans nb-own indice inv-crt
```

Fonction 126 : txt-ob-rel

objectif :

Créer les valeurs spéciales dans les tables des objets, des relations et des ikos pour ce qui concerne les textes ainsi que les codes correspondants dans la table des codes

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
char s-cd[3]; /*chaine pour la fonction itoa (code) */
char s-iko[3];/*chaine pour la fonction itoa (iko)  */
```

argument de la fonction :

/

algorithme :

- ecrire une ligne dans la table des relations avec
  cod = 999
  cod-inv = 000
  cd-targ = 999
  typ = 000
  impn = 004
  iko = b
  pos-iko = jauge de la table des ikos
  nb-own = 000
- incrémenter jauge des relations
- écrire une ligne dans la tables des ikos
  bi = 000
  length = 006
  impn = 0
- incrémenter jauge des ikos
- écrire une ligne dans la table des objets
  cd = 999
  fam = 003
  length = 089
  type == 999
  pos-apt = indice de la table des codes
  nb-targ = 001
- incrémenter jauge de la table des objets
- écrire une ligne dans la table des codes avec cd = 999
- incrémenter jauge de la table des codes
- écrire la jauge de la table des codes (-1 par rapport à la valeur actuelle) dans la table des relations dans la zone pos-own

Fonction 127 : Txt-code

objectif :

Le but est de lier chaque entité ayant un attribut de type texte à l'entité définie TXT, par la création de ligne dans la table des codes qui indiqueront les origines du chemin OF

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

```
int nbr-orig-txt; /* nombre d'origines chemin texte */
int i;            /* recherche table des attributs  */
char s-txt[3];    /* chaine pour texte (fct itoa )  */
```

argument de la fonction :

/

algorithme :

- nbr-orig-txt = 0;
- pour chaque attribut dans la table des attributs
  - si le format est 'T'
    - inscrire une ligne dans la table des codes avec pour valeur le code de l'entité contenu dans la table des attributs à cette même ligne
    - incrémenter jauge de la table des codes
      - ajouter 1 au nombre d'origine du chemin texte dans la zone pos-own de la table des relations
- mettre le nombre d'origines dans nb-own pour le texte

Fonction 1211 1221 1231 128 13311 : Cherch-inv

objectif :

Connaissant la position d'un chemin dans la table des rôles, la fonction permet de connaître la position dans la table des rôles du rôle inverse de celui situé dans la position donnée

nom du fichier :

tables-int.c

variables globales :

/

variables locales :

/

arguments de la fonction :

role_crt (entrée) indice de la ligne traitée dans la table des rôles
inv_crt (sortie) renvoit la position du rôle inverse de celui situé en position role-crt

algorithme :

- inv_crt = 0;
- tant que le r-code à l'indice role-crt de la table des rôles est différent du r-code-inv à l'indice role-crt de la table des rôles
  - incrémenter inv-crt

Fonction 13 : Génér_fich

objectif :

Ce module permet de créer l'environnement pour le programmeur c'est-à-dire un ensemble de structures de données correspondant aux éléments du schéma ainsi qu'un ensemble de définitions

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

/

argument de la fonction :

/

algorithme :

- génération du fichier des "define" comprenant une ligne "define" par entité et par path-name
- génération du fichier reprenant 2 structures par entité et contenant les attributs de cette entité
- génération du fichier reprenant pour chaque entité, le codage permettant sa création lors de l'exploitation du schéma défini

Fonction 131 : Fich_def

objectif :

Le but de cette fonction est de créer un fichier séquentiel contenant une ligne "define" pour chaque nom d'entité et chaque path-name

nom du fichier :

gener-fich.c

variables gobales :

/

variables locales :

```
int i; /* recherche en table des entités
          puis des roles */
```

argument de la fonction :

/

algorithme:

- ouvrir un fichier de nom standard
- pour chaque élément de la table des entités
  - créer une ligne associant par un define le code de l'entité à son nom (fonction creer)
- pour chaque élément de la table des rôles
  - créer une ligne associant par un define le code du rôle à chaque path-name existant (fonction creer)
- fermeture du fichier de nom standard

fonction annexe:

creer(nom,code);
nom : nom de l'entité ou du chemin
code : code en 3 chiffres de l'entité ou du rôle

Fonction 132 : fich-struct

objectif :

Ce module permet de créer deux types de structures par entité. Ces structures participent à la constitution de l'environnement du programmeur.

Chacune de ces deux structures reprennent le nom de l'entité et les noms et les longueurs des attributs de celle-ci.

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *a[43];    /* chaine pour fprintf                  */
int i;          /* recherche en table des entités       */
int j;          /* recherche en table des attributs     */
int it_existe;  /* voir s'il existe au moins un
                   attribut */
```

argument de la fonction :

/

algorithme :

- ouverture de deux fichiers de nom standard
- pour chaque entité définie dans la table des entités
    - créer une première ligne standard dans chaque fichier (fonctions def1 et def3)
    - it-existe = 0;
    - tant que le code de l'entité dans la table des entité est égal au code de l'entité dans la table des attributs
        - pour chaque ligne de la table des attributs créer une ligne dans chacun des deux fichiers et it-existe=1
    - si it-existe = 0 mettre un item technique dans les deux fichiers
    - créer une ligne standard dans chaque fichier (fonctions def2 et def4)
- fermeture des deux fichiers

fonctions annexes:

cr_def1(nom)
nom : nom de l'entité ou du rôle
objectif
Permet pour chaque entité ou rôle de créer une ligne
du type typedef struct i_<nom>_struct {

cr_def2(nom)
nom : nom de l'entité ou du rôle
objectif
Permet de créer une ligne du type } I_<nom>;

cr_def3(nom)
nom : nom de l'entité ou du rôle
objectif
Permet de créer des lignes du type suivant
typedef struct ref_<nom>_str {
int REF,RTYPE,ILEN;

cr_def4(nom)
nom : nom de l'entité ou du rôle
objectif
Permet de créer une lign,e du type } R_<nom>;

Fonction 133 : Fich-creat

objectif :

Cette fonction permet de créer un fichier contenant une suite de strcutures standards et pour chaque entité du schéma le codage nécessaire à la création d'objet de ce type. Cette fonction contribue à la création de l'environnement programmeur.

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char d[20];    /* zone tampon pour include          */
int bool-txt;  /* booleen pour voir s'il y a        */
               /* un attribut de type texte         */
int i;         /* recherche table des entités       */
int j;         /* recherche table des attributs     */
char nom-or[31];/* nom de l'entité                  */
char role-or[31];/* nom du role                     */
```

argument de la fonction :

/

algorithme :

- ouvrir fichier
- copier deux lignes d'includes
- pour chaque entité définie dans la table des entités
  - bool-txt = 0
  - parcourir dans la table des attributs tous les attributs de même e-code
    - si un des attributs à le format T mettre bool-texte à 1
  - si la famille de l'objet est 2
    - chercher les valeurs des entités origines des chemins obligatoires et des deux chemins d'accès
      - si bool-texte != T appel à cr_ref2 sinon appel à cr_txt2
  - si la famille de l'objet est 1
    - chercher les valeurs de l'entité origine et du chemin d'accès
      - si bool-texte != T appel à cr_ref1 sinon appel à cr_txt1
  - si la famille de l'objet est 1
    - si bool-texte != T appel à cr_ref0 sinon appel à cr_txt0
- fermer le fichier

fonction annexe:

```
majmin(a,b)
char *a,*b;
```

objectif

Cette fonction permet de transformer une chaîne dont les caractères sont en majuscules en une chaîne dont les caractères sont en minuscules.

Fonction 1331 : Un-path

objectif :

Trouver dans les tables les valeurs de nom de l'entité et du chemin pour un chemin obligatoire one-to-many dont on connaît l'entité côté cible obligatoire.

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
int indice-trouve /* indice dans les rôles  */
int nbr-orig;     /* nombre d'origines      */
int inverse;      /* indice role inverse    */
int trouve-role;  /* booleen indique si le  */
                  /* role existe            */
char sauv-ent[4]; /* code de l'entité       */
int i;            /* recherche en table     */
```

arguments de la fonction :

e-code (entrée) code de l'entité cible du chemin obligatoire
nom-orig (sortie) nom de l'entité origine du chemin obligatoire
role-orig (sortie) nom du chemin
indice-trouve (sortie) indice de la recherche dans la table des rôles
debut-rech (entrée) indique où doit commencer la recherche dans la table

algorithme :

```
- indice-trouve = debut-rech;
- trouve-role = 0;
- tant que
  le e-code-m est != e-code et que trouve-role =0
    - si e-code-m et e-code sont égaux
          - chercher le role inverse (cherch_inv)
          - si min-con de cet inverse est 1 alors
                trouve-role = 1
            sinon indice-trouve = indice-trouve + 1
      sinon indice-trouve = indice-trouve + 1;
- role-orig = nom du path-name à l'indice-trouve
- nbr-orig=0;
- si dans la table des ent-rol plusieurs e-code-o
     correspondent au r-code de l'indice-trouve
     nbr-orig = nbr-orig + 1 pour chacun d'eux
     si e-code-o = EVE alors nbr-orig = 99;
- si nbr-orig > 1
     alors nom-orig = "RECORD"
  sinon nom-orig = nom correspondant à l'entité dont
  le code est e-code-o
```

Fonction 1333 : cr_ref0

objectif :

Définir une structure permettant la création de l'objet de type "nom" qui n'est lié à aucune autre entité obligatoiremen

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c1[42];  /* fonction fprintf */
char *c3[52];
char *c4[25];
char *c5[71];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

argument de la fonction

nom : nom de l'entité

algorithme :

- génération séquentielle des lignes dans le fichier

Fonction 1334 : cr_ref1

objectif :

Définir une structure permettant la création de l'objet de type "nom" qui est lié à une autre entité par un chemin obligatoire

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c3[52]; /* fonction fprintf */
char *c41[44];
char *c42[25];
char *c5[71];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

arguments de la fonction :

nom : nom de l'entité
nom2 : nom de la deuxième entité
role2 : nom du role

algorithme :

- générer séquentiellement les valeurs dans le fichier

Fonction 1335 : cr_ref2

objectif :

Définir une structure permettant la création de l'objet de type "nom" qui est lié à deux autres entités par deux chemins obligatoires

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c1[70];  /* fonction fprintf */
char *c41[44];
char *c42[25];
char *c5[71];
char *c6[18];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

arguments de la fonction :

nom : nom de l'entité
nom2 : nom de la deuxième entité
nom3 : nom de la troisième entité
role2 : nom du role
role3 : nom du deuxième role

algorithme :

- générer séquentiellement les valeurs dans le fichier

Fonction 1336 : cr_txt0

objectif

Définir une structure permettant la création de l'objet de type "nom" qui n'est lié à aucune autre entité obligatoirement mais qui a un attribut de type "texte"

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c1[48]; /* fonction fprintf  */
char *c3[64];
char *c4[71];
char *c6[44];
char *c7[25];
char *c12[56];
char *c15[31];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

argument de la fonction :

nom : nom de l'entité

algorithme :

- générer séquentiellement les valeurs dans le fichier

Fonction 1337 : cr_txt1

objectif :

Définir une structure permettant la création de l'objet de type "nom" qui est lié à une autre entité par un chemin obligatoire et un attribut de l'entité est de type texte

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c3[64]; /* fonction fprintf */
char *c6[44];
char *c7[25];
char *c8[71];
char *c12[56];
char *c14[48];
char *c15[31];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

arguments de la fonction :

nom : nom de l'entité
nom2 : nom de la deuxième entité
role2 : nom du role

alghorithme :

- générer séquentiellemnt les valeurs dans le fichier

Fonction 1338 : cr_txt2

objectif :

Définir une structure permettant la création de l'objet de type "nom" qui est lié à deux autres entités par deux chemins obligatoires et un attribut est de type texte

nom du fichier :

gener-fich.c

variables globales :

/

variables locales :

```
char *c3[78]; /* fonction fprintf */
char *c4[71];
char *c6[44];
char *c7[25];
char *c8[18];
char *c12[56];
char *c14[48];
char *c15[31];
char *nom-ent[31]; /* transformer le nom de lettres
                      majuscules en minuscules */
```

arguments de la fonction :

nom : nom de l'entité

algorithme :

- générer séquentiellement les valeurs dans le fichier

Fonction 14 : Creat-base

objectif :

Lorsque les tables internes et externes sont créées, il faut les introduire dans une base de données pour permettre son exploitation. Les tables internes peuvent soit remplacer les anciennes tables d'une base de données, soit constituer la première étape de l'exploitation d'une nouvelle base de données.

nom du fichier :

creat_base.c

variables globales :

```
define sys "000"          /* appel bd */
define object 0
define text 3
define of 4
struct m5_struct z_val;
int z_recref;
int z_oref;
int z_pathlist[10];
int z_curlist[10];
```

variables locales :

```
struct dbstat_struct dbstat1;
struct intercd;          /* code sous forme caractères */
struct interiko;         /* iko sous forme caractères  */
int opcode;              /*  appel bd */
int getcode;
int keycode;
int operator;
int prem-obj;            /* un indice par table        */
int prem-relations;
int prem-cd;
int prem-iko;
int pre-ent;
int pre-role;
int pre-attr;
int ent-rol;
char name[6];            /* nom de la base de données  */
char s-cd[3];            /* fonction itoa  (codes)     */
char s-iko[3];           /* fonction itoa (iko)        */
```

argument de la fonction :

/

algorithme :

- saisir le nom de la base de données pour y mettre les tables
- ouvrir cette base de données
  - si cette base existe déjà
    - supprimer les anciennes tables
  - si cette base n'existe pas encore
    - créer les nouveaux fichiers d'une base
    - ouvrir cette nouvelle base
- création de deux variables de références
- création de l'article système permettant l'exploitation des tables internes
- chargement de chacune des 8 tables
- fermeture de la base de données

Fonction 141 : Nouveaux fichiers

objectif

Le but de ce module est de créer un ensemble de fichiers DBC (DBASE3 du LATTICE C) qui correspondent à la structure de l'atelier

nom du fichier :

nouv-fich.c

variables globales :

4 structures de fichiers DBC

variables locales :

```
char *inter[12]; /* zone de transfert                    */
char *inter2[12];
char *o-c-id;    /* un pointeur par fichier d'index    */
char *o-typ;
char *a-c-id;
char *o_t_seq;
char *l-c-id;
char *m-typ1;
char *m-typ2;
char *t-c-id;
char *o-t-g-s;
```

argument de la fonction :

nom : chaîne de 5 caractères donnant le nom de la base

algorithme

- créer le nom du fichier des objets de type 0
- créer le fichier
- créer le nom du premier fichier d'index
- créer le premier fichier d'index
- créer le nom du deuxième fichier d'index
- créer le deuxième fichier d'index
- créer le nom du fichier des objets de type 1
- créer le fichier
- créer le nom du premier fichier d'index
- créer le premier fichier d'index
- créer le nom du deuxième fichier d'index
- créer le deuxième fichier d'index
- créer le nom du fichier des objets de type 2
- créer le fichier
- créer le nom du premier fichier d'index
- créer le premier fichier d'index
- créer le nom du deuxième fichier d'index
- créer le deuxième fichier d'index
- créer le nom du troisième fichier d'index
- créer le troisième fichier d'index
- créer le nom du fichier des objets de type texte
- créer le fichier
- créer le nom du premier fichier d'index
- créer le premier fichier d'index
- créer le nom du deuxième fichier d'index
- créer le deuxième fichier d'index

Fonction 15 : Visualisation

objectif :

Lorsque les tables externes et internes sont créées, cette fonction permet de visualiser leurs contenus à l'écran.

nom du fichier :

visualisation.c

variables globales :

/

variables locales :

```
int pre-ent;   /* un indice par table */
int pre-role;
int pre-attribute;
int pre-ent-rol;
int pre-iko;
int pre-cd;
int pre-objects;
int pre-relations;
```

argument de la fonction :

/

algorithme :

```
- pour chaque ligne de la table des entités
      afficher le contenu des différentes zones
- pour chaque ligne de la table des roles
      afficher le contenu des différentes zones
- pour chaque ligne de la table des attributs
      afficher le contenu des différentes zones
- pour chaque ligne de la table des ent_role
      afficher le contenu des différentes zones
- pour chaque ligne de la table des objets
      afficher le contenu des différentes zones
- pour chaque ligne de la table des relations
      afficher le contenu des différentes zones
- pour chaque ligne de la table des ikos
      afficher le contenu des différentes zones
- pour chaque ligne de la table des codes
      afficher le contenu des différentes zones
```

Fonction 90 : Transf2

objectif :

Cette fonction permet de transférer une chaîne de caractères dans une autre en ajoutant des 0 devant la chaîne de caractères à concurrence de la longueur de la chaîne réceptrice.

nom du fichier :

transf2.c

variables globales :

/

variables locales :

```
int i;
int j;
int r;
```

arguments de la fonction :

b1 (sortie) chaîne de caractères réceptrice
b2 (entrée) chaîne de caractères émettrice
length (entrée) longueur de la chaîne émettrice

algorithme :

- r=longueur de b2;
- mettre '0' pour la chaîne b1 de la position 0 jusqu'à length - r
- copier la chaîne;
- mettre dans b1 le caractère de fin de chaîne

Fonction 91 : Itoa

objectif :

Cette fonction permet de changer un entier en une chaîne de caractères

nom du fichier :

itoa.c

variables globales :

/

variables locales :

```
int i;
```

arguments de la fonction :

n (entrée) entier à transformer
s (sortie) chaîne de caractères

algorithme :

- i = 0;
- générer la chaîne de caratère
- inverser la chaîne

fonctions annexes :

reverse(s)
s : chaîne de caractères

strlen(s)
s : chaîne de caractères